Raisonner sur des données en agroécologie : application à la sélection d'espèces végétales de service

Elie Najm

THÈSE POUR OBTENIR LE GRADE DE DOCTEUR DE L'UNIVERSITÉ DE MONTPELLIER

En Informatique

École doctorale : Information, Structures, Systèmes (I2S)

Unité de recherche : LIRMM

Raisonner sur des données en agroécologie : application à la sélection d'espèces végétales de service

Présentée par Elie NAJM
le 13 décembre 2022

Sous la direction de Marie-Laure MUGNIER
et Christian GARY

Devant le jury composé de

Marie-Laure MUGNIER, Professeur des Universités, Université de Montpellier	Co-directrice de thèse
Christian GARY, Directeur de recherche, Inrae Montpellier	Co-directeur de thèse
Jacques-Eric BERGEZ, Directeur de recherche, Inrae Toulouse	Rapporteur
Fatiha SAÏS, Professeur des Universités, Université Paris Saclay	Rapportrice
Marianne HUCHARD, Professeur des Universités, Université de Montpellier	Examinatrice
Kamal KANSOU, Chargé de recherche, Inrae Nantes	Examinateur
Jean-François BAGET, Chargé de Recherche, Inria	Invité
Raphaël MÉTRAL, Ingénieur d'études, Inrae Montpellier	Invité

UNIVERSITÉ DE MONTPELLIER

Table des matières

CHAPITRE 1

Introduction

Au cours du 20ème siècle, et plus particulièrement après la seconde guerre mondiale, l'agriculture a évolué vers des formes intensives qui ont conduit à une augmentation rapide de la productivité de la terre et du travail. Durant la période de 1960 à 1990, la *révolution verte* a permis de mettre à disposition de nombreux pays un approvisionnement diversifié et économiquement abordable, entrainant une réduction importante des risques de disette et de famine à travers le monde.

Cependant, cette nouvelle forme d'agriculture dite "moderne" a eu de graves conséquences sur l'environnement et la santé humaine [HJJ22]. La modernisation de l'agriculture est passée par l'utilisation massive de véhicules agricoles motorisés, comme les tracteurs, qui ont conduit à une réorganisation des parcelles et à une modification des paysages, passant ainsi de parcelles entourées d'arbres et de haies, favorisant la présence de nombreux oiseaux, insectes et petits mammifères, à des parcelles de monoculture, c'est-à-dire de grandes surfaces de cultures uniques, plus adaptées au passage des véhicules agricoles motorisés, mais réduisant les habitats naturels favorables à la biodiversité et à la régulation naturelle des maladies et de ravageurs des cultures. Dès lors, le maintien, voir l'accroissement des rendements, est passé par une utilisation massive de produits phytosanitaires afin de lutter contre les maladies et les ravageurs se propageant désormais plus facilement. Cependant, l'épandage de ces produits phytosanitaires a de graves conséquences sur de nombres espèces naturelles touchées, alors même qu'elles n'en étaient pas la cible. En effet, ces substances peuvent fortement perturber la reproduction et conduire à une mortalité accrue de certains animaux

comme les pollinisateurs. Elles conduisent également à l'élimination des décomposeurs (essentiellement des bactéries et champignons), indispensables à la décomposition de la matière organique du sol et nécessaires à la fertilité du sol. Les pesticides sont également suspectés d'augmenter le risque de développement de certaines maladies, comme certains types de cancer, chez les populations exposées : agriculteurs, ouvriers des usines de conditionnement de pesticides, riverains des exploitations, consommateurs des produits agricoles, etc. Par ailleurs, la recherche de rendements accrus a conduit à une utilisation massive de fertilisants (ou engrais) chimiques. Ainsi, les engrais azotés, qui ne sont qu'en partie absorbés par les plantes, conduisent à une pollution polymorphe : contamination de l'eau par les nitrates et émission dans l'atmosphère de protoxyde d'azote, un puissant gaz à effet de serre, ainsi que d'ammoniac, à l'origine de nombreux pics de pollution en période d'épandage. Nous ne citons là que les plus visibles des conséquences de l'utilisation massive de produits de synthèse. Plus généralement, il est maintenant avéré que l'agriculture intensive conduit à une pollution multiforme de l'environnement (sols, eau et air), qu'elle a des impacts négatifs sur la santé humaine, et qu'elle contribue significativement à l'appauvrissement des sols, à la baisse de biodiversité et au changement climatique à l'échelle planétaire.

Par opposition à une agriculture basée sur la monoculture et l'utilisation intensive d'intrants de synthèse, l'*agroécologie*, née de l'application de l'écologie à l'agriculture, place au centre la notion d'agroécosystème [Wez+09]. Un *agroécosystème* est un écosystème modifié par l'homme en vue de produire des aliments et d'autres biens consommables : un tel système de production s'appuie sur les fonctionnalités offertes par les êtres vivants (faune et flore) et éléments non vivants (eau, air, matières solides, etc), ainsi que sur les interactions possibles entre ces différentes entités.

Selon cette nouvelle approche de l'agricuture, les systèmes agricoles ont pour objectif de produire non seulement des biens consommables (aliments, énergie, fibres, matières premières, . . .), mais également des *services écosystémiques* [Les+15b]. Par services écosystémiques, on entend les services rendus par les écosystèmes aux sociétés humaines. Autrement dit, la parcelle agricole n'est plus seulement vue comme un espace de production d'une culture principale (et souvent unique), mais comme un écosystème à même de rendre différents services. Ces services écosystémiques sont non seulement des services attendus par la société (comme le maintien ou la restauration de la qualité de l'environnement, la préservation de certains paysages, etc.) mais ils sont également utiles à l'agriculture (comme les

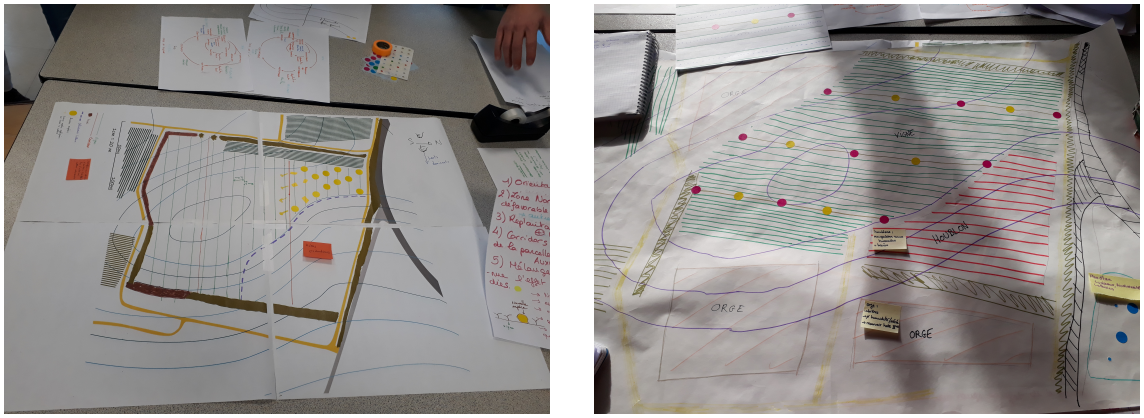


FIGURE 1.1 – Images issues d'un atelier de conception ayant pour but de sélectionner des espèces à implanter dans un système viticole en vue de fournir certains services écosystémiques, en particulier la régulation des bioagresseurs. Dans le cadre du projet Casdar *AgroEcoPérenne* et du projet Plan National Dépérissement du Vignoble *Tradevi*.

services de régulation des bioagresseurs, d'entretien de la qualité et de la fertilité des sols, de fourniture d'éléments nutritifs aux plantes cultivées, etc.).

Il est maintenant établi que la restauration durable de compromis entre production agricole et services écosystémiques passe par la réintroduction de biodiversité aux échelles des parcelles agricoles [Dur+15]. Cela conduit à des agroécosystèmes plus complexes, comprenant un plus grand nombre de composantes et d'interactions entre elles. Afin de concevoir ces nouveaux systèmes de culture, les chercheurs en agronomie s'appuient sur des *ateliers de conception* réunissant plusieurs experts de domaines différents [Sim+17]. En effet, l'activité de conception nécessite d'intégrer des connaissances de divers domaines : l'agronomie, en ce qui concerne les fonctions et la gestion des agroécosystèmes, l'écologie, en ce qui concerne les relations entre les organismes vivants et leur environnement, ainsi que d'autres domaines d'expertise, comme les sciences de gestion, et des savoir-faire agricoles issus des agriculteurs. Cependant, ces ateliers de conception sont très coûteux en temps. En outre, ils ne permettent le développement de tels systèmes que pour des cas spécifiques dans des contextes particuliers (comme le cas de la régulation des bioagresseurs sur une parcelle de vigne située à un endroit précis, cf. figure 1.1).Une autre limitation de ces ateliers est qu'ils ne peuvent considérer que des espèces de plantes déjà connues par les experts réunis.

Ceci fait apparaître le besoin crucial d'outils informatiques aidant au choix des espèces composant ces nouveaux agroécosystèmes. L'augmentation de la biodiversité peut être réalisée soit en associant plusieurs espèces agricoles dans les mêmes parcelles, soit en installant des espèces dites *de service* qui ne fournissent

FIGURE 1.2 – Différents exemples de cultures de service semées en interligne dans les vignobles. De gauche à droite : *Hordeum vulgare*, *Phacelia tanacetifolia*, et *Vicia villosa*. Photos fournies par Alexis Thoumazeau, Léo Garcia et Yvan Bouisson.

pas directement des biens consommables mais des services écosystémiques. C'est à la problématique d'*aide au choix d'espèces de service* que nous nous intéressons dans cette thèse.

Bien que notre démarche se veuille suffisamment générale pour être applicable à divers cas d'étude, nous la développons sur un cas particulier, celui de *l'enherbement viticole*, qui consiste à installer dans les vignobles une végétation herbacée de service. Les enherbements viticoles prennent une grande variété de formes (voir figure 1.2). Ils peuvent se développer spontanément et reposer sur des espèces natives ou bien être semés avec une seule espèce ou un mélange prédéterminé d'espèces. Leur développement dépend des conditions de sol et de climat, mais également des pratiques culturales (fauche, destruction partielle ou totale). Ces enherbements peuvent contribuer positivement ou négativement à la production de différents services écosystémiques [Gar+18].

Du point de vue informatique, notre démarche consiste à associer d'une part des connaissances scientifiques (ou "expertes") formalisées dans un langage permettant des raisonnements, et d'autre part des données, qui ont été collectées de façon indépendante de notre cas d'étude. En effet, des travaux scientifiques récents en (agro)écologie ont permis d'identifier des relations entre les *traits fonctionnels* des plantes et les *fonctions* écosystémiques qu'elles fournissent ; ces fonctions contribuant elles-mêmes à la production de *services* écosystémiques [Gar+19 ; Gar+20 ; Dam+15]. Les traits fonctionnels sont des caractéristiques mesurables des espèces végétales (à savoir, morphologiques, physiologiques ou phénologiques) qui sont pertinentes pour étudier la réponse de ces organismes aux perturbations environnementales et/ou leurs effets sur les propriétés de l'écosystème [Vio+07 ; GN11]. Les fonctions écosystémiques sont souvent associées à des combinaisons de traits, qui peuvent chacun contribuer positivement ou négativement à la fonction. Par exemple, plusieurs traits fonctionnels du système racinaire d'une plante peuvent être associés positivement à la fonction de stabilité structu-

rale du sol, qui soutient le service de maintien de la qualité du sol [Gar+19].

Cependant, aucun outil ne permet actuellement la représentation des liens multiples entre des traits fonctionnels, qui peuvent être en nombre très important, et les différentes fonctions à la base d'une gamme de services écosystémiques. Il ne s'agit pas seulement de représenter ces liens, mais d'en tirer ensuite des conclusions utiles à la sélection et à l'assemblage d'espèces, et ce, dans une démarche de conception de systèmes agroécologiques. Par ailleurs, la communauté des écologues a compilé dans la base de données TRY [Fra20] des valeurs de traits fonctionnels recueillies sur un grand nombre d'espèces végétales dans une large gamme de conditions [1]. C'est une source d'information très riche qui n'a pas encore été valorisée pour le choix d'espèces de service en agriculture. Partant de ce constat, notre hypothèse de départ a été la suivante : dès lors qu'à des services et fonctions attendus seraient associées des valeurs de traits fonctionnels, des bases de données telles que TRY devraient permettre d'identifier les espèces favorables à ces fonctions et services. Plus généralement, notre fil directeur a été la question suivante :

Peut-on exploiter les données disponibles sur les traits fonctionnels des plantes et les combiner à une représentation adéquate des connaissances scientifiques sur les relations traits-fonctions-services pour évaluer la contribution potentielle d'une espèce végétale quelconque à un service écosystémique visé ?

Il nous faut donc intégrer des données, qui proviennent possiblement de plusieurs sources, et des connaissances expertes, les deux étant susceptibles d'évoluer. Pour réaliser cette intégration, nous considérons des techniques à la croisée de deux domaines : la représentation de connaissances et les raisonnements, un domaine historiquement au coeur de l'intelligence artificielle, et la gestion de données. Plus précisément, nous nous appuyons sur un paradigme récent, connu sous le nom de *Ontology-Based Data Access* [Pog+08 ; Len18] (accès aux données basé sur une ontologie, OBDA) qui propose une architecture en trois niveaux :

- les sources de données, qui peuvent préexister indépendamment de l'application visée ;
- les connaissances mises en oeuvre dans l'application, qui sont structurées autour d'une ontologie et représentées dans un langage formel permettant les raisonnements automatisés ;
- et les mappings qui lient données et connaissances, permettant de sélectionner les données pertinentes et de les traduire en utilisant le vocabulaire de l'ontologie.

1. https://www.try-db.org/TryWeb/Database.php

En d'autres termes, on a une base de connaissances, composée d'une part d'une ontologie, et d'autre part d'une base de faits qui est définie par les mappings qui la lient aux données. Les requêtes au système sont formulées en utilisant le vocabulaire ontologique, ce qui permet à un utilisateur (agronome) de faire abstraction du codage spécifique des données, et leur évaluation repose sur les raisonnements associés aux connaissances, ce qui permet de prendre en compte des informations inférées qui n'étaient pas directement productibles à partir des données.

Le paradigme OBDA a suscité beaucoup d'intérêt à partir des années 2010. Plusieurs implémentations sont désormais disponibles, des systèmes matures [Cal+11 ; Cal+17] aux prototypes de recherche [SAM14 ; Bur+20b], tous basés sur des langages du web sémantique, à savoir OWL2 QL, ou RDF Schema et de légères extensions de celui-ci. Alors que OBDA a été déployé dans diverses entreprises et organisations publiques [Cal+11 ; Civ+13 ; Kha+17 ; Kha+18 ; Kal+20], il n'a apparemment pas encore été appliqué à l'agriculture/agroécologie. D'autre part, ce cadre a été mis en oeuvre jusqu'ici avec des langages d'ontologie très peu expressifs qui ne sont pas suffisants pour les connaissances que nous souhaitons mettre en représenter.

Précisons que notre objectif n'est pas de concevoir un outil "presse-bouton" qui délivrerait une recommandation en fonction d'un certain nombre de paramètres d'entrée, sans possibilité d'accès aux calculs de cette recommandation, mais plutôt un outil capable de proposer des alternatives justifiées en rendant transparentes les étapes de son raisonnement. Il ne s'agit pas non plus de prendre une décision pour l'utilisateur mais plutôt de l'accompagner dans sa démarche de conception en ouvrant le "champ des possibles".

Finalement, notre objectif a été de concevoir (et implémenter) un outil qui satisfasse les exigences suivantes :

1. intégrer des données, qui peuvent provenir de sources hétérogènes et indépendantes,
2. combiner les données avec une représentation formelle de la connaissance experte qui permet le raisonnement,
3. prendre en compte les raisonnements lors du calcul des réponses à une requête,
4. permettre à un utilisateur final de formuler des requêtes dans un vocabulaire familier,

5. adopter une démarche suffisamment générique pour être applicable à d'autres cas d'étude suivant l'approche trait-fonction-service,
6. permettre une évolution facile en termes de sources de données (intégration simplifiée de nouvelles bases de données) et de connaissances expertes (les connaissances en agroécologie étant susceptibles d'évoluer rapidement puisqu'elles sont en plein essor, il faut faciliter l'évolution des connaissances expertes),
7. permettre l'explication des résultats à l'utilisateur, en lui rendant intelligibles les étapes du raisonnement ayant conduit à ces résultats.

Nous présentons le développement de cet outil pour le cas d'étude de l'enherbement des vignes. L'implémentation a été effectuée dans le logiciel Graal [Bag+15] dédié à l'interrogation de bases de connaissances composées de règles logiques, appelées règles existentielles (un formalisme qui inclut les règles que nous utilisons) et développé par l'équipe BOREAL (Inria & LIRMM) [2].

Organisation du manuscrit

La suite du manuscrit est organisée de la façon suivante.

Chapitre 2 Ce chapitre a pour objectif de situer notre travail, d'une part dans le domaine de la représentation des connaissances et des raisonnements en lien avec l'exploitation de données, d'autre part dans le cadre de l'ingénierie agroécologique. Après quelques rappels sur la logique du premier ordre et l'introduction de notions fondamentales sur les bases de connaissances et requêtes, la section 2.1 présente les logiques de description, les langages du web sémantique et les langages à base de règles. Elle aborde finalement le sujet de l'accès à des données en provenance de différentes sources et présente le paradigme Ontology-Based Data Access (OBDA) ainsi que les systèmes existants implémentant ce paradigme. La section 2.2 présente d'une part les ressources sémantiques en agroécologie qui nous paraissent pertinentes (thésaurus, ontologies, portails, outils logiciels), d'autre part les outils existants d'aide à la sélection d'espèces, dont on montre qu'ils ne correspondent pas à nos objectifs.

Chapitre 3 Dans ce chapitre, nous définissons le cadre formel informatique sur lequel notre modélisation est basée. Nous présentons d'abord l'architecture gé-

2. Version courante (le logiciel étant devenu InteGraal) : https://gitlab.inria.fr/rules/integraal.

nérale de notre outil, qui est inspirée de celle d'OBDA (section 3.1), le langage de représentation de connaissances choisi (section 3.2) et les différents types de mappings que nous avons définis. En bref, le langage de représentation de connaissances choisi est constitué de règles Datalog avec négation stratifiée, auxquelles nous avons ajouté des prédicats et fonctions calculés, ce qui a nécessité de définir un système de typage des valeurs. Nous définissons différents types de mappings, qui vont permettre de nettoyer, agréger puis transformer les données afin de les traduire en des faits de plus haut niveau utilisant le vocabulaire de l'ontologie.

Les deux chapitres suivants constituent le coeur de notre contribution.

Chapitre 4 Ce chapitre présente la construction des *faits* de la base de connaissances. Une partie des faits est obtenue à partir des sources de données et l'autre à partir de connaissances expertes. La section 4.1 présente les bases de données intégrées dans notre système et les différentes étapes permettant l'acquisition de faits à partir de ces bases de données. La section 4.2 présente une méthodologie d'acquisition des connaissances expertes sous forme de diagrammes liant traits fonctionnels, fonctions et services écosystémiques, et établissant des correspondances entre les traits fonctionnels des diagrammes et des identifiants dans les bases de données.

Chapitre 5 Ce chapitre présente d'abord la base de *règles* et les différents types de règles qui la composent. Ici, on distingue les règles traduisant l'ontologie de domaine, des règles permettant d'effectuer des calculs (et utilisant un vocabulaire qui n'a pas forcément de sens pour un utilisateur) de façon à consolider les valeurs de traits issues des données, ainsi que calculer des valeurs de fonctions et services en intégrant une notion de fiabilité liée aux valeurs manquantes. Ce chapitre présente également en dernière section (section 5.3) une *évaluation* des résultats de l'outil. Cette évaluation est faite en comparant les résultats obtenus par l'outil à ceux issus de la littérature scientifique. Nous observons une bonne corrélation entre ces deux résultats et surtout, nous notons l'importance du paramètre de fiabilité dans notre étude : plus la valeur de fiabilité est élevée (autrement dit, moins il y a de données manquantes), plus la corrélation entre les résultats est bonne.

Chapitre 6 Dans ce chapitre, nous nous intéressons à expliquer un fait inféré par le système. Plus précisément, nous cherchons à fournir des *explications pertinentes*

pour l'utilisateur, dans le sens où une explication fournie utilise un vocabulaire qui lui est compréhensible. Pour cela, nous fixons en amont le sous-ensemble du vocabulaire que les explications données doivent utiliser. Nous présentons un cadre théorique, non encore implémenté, afin de produire ce type d'explications.

Enfin, le **chapitre 7** conclut cette étude et dresse des perspectives.

CHAPITRE 2

Accès aux données basé sur des connaissances

Dans ce chapitre, nous donnons d'abord une vue d'ensemble des principaux langages de représentation de connaissances utilisés dans le cadre de l'accès aux données. Ceci permet de situer nos travaux dans le domaine informatique de la représentation des connaissances et des raisonnements. Puis nous faisons un tour d'horizon des ressources sémantiques en agronomie et en agroécologie pertinentes par rapport à notre cas d'étude, ainsi que des applications d'aide à la sélection de plantes. Ceci permet de situer notre approche par rapport aux outils existants.

2.1 Bases de connaissances et accès aux données

Une *base de connaissances* est un ensemble d'assertions formelles qui représentent des connaissances spécifiques à un domaine donné. Les connaissances peuvent être représentées en utilisant différents langages formels ayant chacun leur propre syntaxe. Cependant, tous ces langages ont généralement une sémantique qui se traduit en logique du premier ordre. On peut donc voir, au niveau formel, une base de connaissances comme un ensemble de formules logiques. C'est pourquoi nous commencerons cette partie par quelques rappels de logique du premier ordre.

2.1.1 Logique du premier ordre

La logique du premier ordre admet des formules qui sont construites à partir d'un vocabulaire. Ce vocabulaire est constitué :

- d'un ensemble de (noms de) *prédicats* ; à chaque prédicat est associé un entier (positif ou nul), appelé son *arité*, qui spécifie son nombre d'arguments ;
- d'un ensemble de *constantes* et d'un ensemble de *variables*, disjoint du premier ;[1]
- éventuellement (mais nous ne le considérons pas dans ce chapitre), d'un ensemble de noms de *fonctions*, chacune munie de son arité.

Les objets apparaissant dans les formules sont appelés des *termes*. Un terme peut être une constante ou une variable, auquel cas c'est un terme simple, ou bien être construit inductivement à partir de noms de fonctions et de termes simples, auquel cas c'est un terme complexe. Dans ce chapitre, nous ne considérons pas les termes complexes.

Un *atome* est de la forme $p(t_1, \ldots, t_k)$ où p est un prédicat d'arité k et les t_i des termes. Par exemple, l'atome *estMèreDe(alice,bob)* peut formaliser le fait que "alice est la mère de bob".

Un prédicat p peut être propre à un vocabulaire donné, ou bien être prédéfini et commun à tous les vocabulaires, comme les comparateurs arithmétiques $<, =$, etc. Une *formule* sur un vocabulaire se définit par induction, à partir des atomes, des connecteurs logiques et des quantificateurs :

1. Un atome $p(t_1, \ldots, t_n)$ est une formule (atomique). Pour simplifier l'écriture, on note $t_1 = t_2$ au lieu de $= (t_1, t_2)$ et similairement pour les autres prédicats de comparaison.
2. Si A est une formule, alors $\neg A$ est une formule.
3. Si A et B sont des formules, alors $(A \wedge B), (A \vee B), (A \rightarrow B)$ sont des formules.
4. Si A est une formule et X une variable de A, alors $\forall X\ A$ et $\exists X\ A$ sont des formules.

Par la suite, on considère une priorité sur les opérateurs, ce qui permet d'omettre certaines parenthèses : $\neg$ est prioritaire sur $\wedge$ et $\vee$, eux-mêmes prioritaires sur $\rightarrow$.

Dans la formule $\forall X\ A$ (ou $\exists X\ A$), X est une variable *quantifiée* et A est la *portée* de la quantification $\forall X\ A$ (ou $\exists X\ A$). Une *occurrence* de la variable X est

1. Dans la suite, les noms des variables commenceront par une majuscule, et les noms de constantes par une minuscule.

liée si elle est dans la portée d'une quantification portant sur X. Sinon, cette occurrence est *libre*. Une *variable* est libre si elle a au moins une occurrence libre. Une formule est dite *fermée* lorsqu'elle n'a *aucune* variable libre. Par exemple, $estMèreDe(alice, X)$ n'est pas fermée tandis que la formule $\exists X\ estMèreDe(alice, X)$ est bien fermée, et veut dire que "alice est mère (d'un enfant) X non identifié". La formule $\forall X\ (Parent(X) \wedge Humain(X) \rightarrow \exists Y\ (estParentDe(X, Y) \wedge Humain(Y)))$ est également une formule fermée et veut dire que "tout parent humain est parent d'un humain".

2.1.1.1 Interprétation et modèle

Une *interprétation* d'un vocabulaire encode un "monde possible", qui donne une signification aux symboles du vocabulaire. Une formule construite sur le vocabulaire pourra être vraie ou fausse dans ce monde. Plus précisément, une interprétation I d'un vocabulaire, est composée d'un ensemble non vide D appelé le domaine de I et d'une définition de la signification des symboles du vocabulaire :

- I associe à chaque constante c un élément de D : $I(c) \in D$.
- I associe à chaque prédicat p d'arité k un ensemble de k-tuples sur D : $I(p) \subseteq D^k$. Dans le cas d'un prédicat d'arité 0 ($p = 0$), on a soit $I(p) = \{()\}$ qui signifie vrai, soit $I(p) = \{\}$ qui signifie faux.

Un atome $p(e_1, \ldots, e_k)$ pour lequel à chaque terme e_i on a affecté un élément d_i de D est vrai pour I si $(d_1, \ldots, d_k) \in I(p)$. Une formule fermée $\forall X\ A$ est vraie pour I, si pour tout élément $d \in D$, la formule $A[d]$ est vraie, où $A[d]$ désigne la formule obtenue à partir de A en affectant à X l'élément d ; de la même façon, une formule $\exists X\ A$ est vraie pour I s'il existe $d \in D$ telle que la formule $A[d]$ est vraie ; $\neg A$ est vraie pour I si A est fausse pour I ; $A \wedge B$ est vraie pour I, si A et B sont vraies pour I ; $A \vee B$ est vraie pour I, si A ou B est vraie I ; $A \rightarrow B$ est vraie pour I si A est fausse ou B est vraie.

Lorsqu'une formule fermée f est *vraie* dans une interprétation I, on dit que I est un *modèle* de f. Une formule (fermée) f est *satisfiable* si elle admet au moins un modèle. On dit qu'une formule f est *conséquence (sémantique)* d'une formule ou d'un ensemble de formules g et on note $g \models f$ si tout modèle de g est un modèle de f (intuitivement : à chaque fois que g est vraie, f l'est aussi). On remarque que si g n'est pas satisfiable, toute formule (fermée) est conséquence de g.

Une interprétation est un modèle pour un ensemble de formules fermées si elle est un modèle pour chacune de ses formules. On étend naturellement les notions de satisfiabilité et de conséquence aux ensembles de formules : si $\mathcal{F}$ est un ensemble de formules (fermées), alors $\mathcal{F}$ est satisfiable s'il admet au moins un

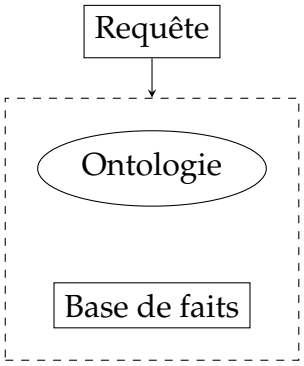


FIGURE 2.1 – Base de connaissances

modèle, et une formule fermée f est conséquence de $\mathcal{F}$ si tout modèle de $\mathcal{F}$ est un modèle de f.

2.1.2 Bases de connaissances

Nous voyons une *base de connaissances* comme une structuration d'un ensemble de formules logiques. En pratique, les éléments d'une base de connaissances peuvent être décrits dans une syntaxe différente de la logique du premier ordre mais sont traduisibles dans cette logique.

Une base de connaissances est généralement composée de deux parties (Figure 2.1 : boîte en pointillés) :

- La *base de faits* qui contient des atomes instanciés (c'est-à-dire dont tous les termes sont des constantes) appelés *faits*. Ces faits décrivent des connaissances relatives à une situation spécifique, par exemple, *estMèreDe(alice,bob)*
- L'*ontologie* qui contient toutes les autres formules et décrit des connaissances plus générales. Suivant le langage considéré, les formules autorisées seront plus ou moins expressives. Selon Gruber [Gru95], une ontologie est une spécification explicite d'une conceptualisation partagée d'un domaine. Elle spécifie dans un langage formel un vocabulaire composé de concepts (comme *Père*, *Mère*, *Parent*, *Enfant*) et de relations pouvant exister entre des instances de ces concepts (comme *aPourParent*, *estMèreDe*, *estPèreDe*), ainsi que des relations sémantiques entre les éléments du vocabulaire (par exemple, *"toute instance de Père est une instance de Parent"*).

Précisons que dans le domaine du web sémantique, on utilise souvent le terme "ontologie" pour ce que nous appelons "base de connaissances", c'est-à-dire que l'ontologie comporte aussi des connaissances factuelles.

Deux problèmes fondamentaux se posent sur les bases de connaissances : d'une part déterminer la consistance ou *satisfiabilité* d'une base de connaissances

(c'est-à-dire la satisfiabilité de l'ensemble des formules qui composent la base de connaissances), d'autre part calculer l'ensemble des *réponses à des requêtes*, la notion de réponse reposant sur la conséquence logique ; pour ce deuxième problème, on suppose que la base de connaissances est satisfiable, sinon, la base de connaissances n'ayant aucun modèle, par vacuité tout est conséquence de la base. De nombreux problèmes de plus haut niveau s'appuient sur ces deux problèmes fondamentaux.

Il existe différentes catégories de requêtes sur une base de connaissances. Nous considérons ici la notion de *requête du premier ordre*, qui est une formule de la logique du premier ordre. Ces requêtes la même expressivité que les requêtes de l'algèbre relationnel, le formalisme sur lequel est basé SQL, le langage d'interrogation des bases de données relationnelles, [AHV94].

Les variables libres (non quantifiées) d'une requête sont nommées *variables réponses*. S'il n'y a aucune variable libre, la requête est *booléenne*. Les requêtes fondamentales considérées pour l'interrogation de bases de données ou de connaissances sont les requêtes conjonctives. Une *requête conjonctive* est une conjonction d'atomes dont les variables peuvent être existentiellement quantifiées ou libres, auquel cas elles sont appelées *variables réponses* (ce sont les variables pour lesquelles on cherche une valeur). Formellement, une requête conjonctive Q est de la forme

$$\exists \overrightarrow{X}\ C(\overrightarrow{X}, \overrightarrow{Y})$$

où $\overrightarrow{X}$ et $\overrightarrow{Y}$ sont des tuples de variables (celles de $\overrightarrow{Y}$ constituant les variables réponses) et C une conjonction d'atomes. On utilise aussi la notation $Q(\overrightarrow{Y})$ pour dire que Q a pour tuple de variables réponses $\overrightarrow{Y}$.

Par exemple, "Trouver tous les parents qui ont une fille" peut s'exprimer par la requête conjonctive suivante : $Q(Y) = \exists X\ (aPourParent(X, Y) \wedge Fille(X))$; et déterminer "s'il existe un parent qui a une fille" est une requête conjonctive booléenne qui s'exprime comme suit : $Q() = \exists X \exists Y\ \ (aPourParent(X, Y) \wedge Fille(X))$.

Soit $\mathcal{K}$ une base de connaissances vue comme un ensemble de formules et $Q(X_1, \ldots, X_k)$ une requête (c'est-à-dire dont les variables réponses sont $X_1, \ldots, X_k$). Le k-uplet de constantes $(a_1, \ldots, a_k)$ est une *réponse* à Q sur $\mathcal{K}$ si $\mathcal{K} \models Q_a$, où Q_a est obtenue à partir de Q en remplaçant chaque X_i par a_i. Dans le cas d'une requête booléenne, si l'ensemble de réponses est égal à $\{()\}$, cela veut dire que la réponse est "vrai" et si cet ensemble est égal à $\{\}$, cela veut dire que la réponse est "faux".

2.1.3 Logiques de description

Les *logiques de description* (LDs) [Baa+03] constituent une famille de langages de représentation de connaissances utilisés pour représenter et raisonner sur les connaissances d'un domaine d'application d'une manière structurée. Elles se sont traditionnellement focalisées sur la représentation des ontologies et les raisonnements associés. Le vocabulaire d'une logique de description est constitué d'un ensemble de prédicats unaires appelés *concepts* (qui représentent des classes d'individus), de prédicats binaires appelés *rôles* (qui représentent des relations entre individus) et de *constantes*. Notons que la syntaxe des LDs est sans variable, même si des variables apparaissent dans leur traduction logique. Différentes LDs existent, chacune étant définie par les *constructeurs* qu'elle autorise, et éventuellement par les restrictions qu'elle impose sur leur emploi. À partir des constructeurs autorisés et du vocabulaire, on peut construire inductivement des concepts et des rôles plus complexes appelés concepts (ou rôles) construits. Ainsi, le concept construit $Parent \sqcap Humain$ représente tous les individus appartenant à la fois au concept $Parent$ et au concept $Humain$; et le concept construit $\exists\, estParentDe.Humain$ représente tous les individus qui sont parents d'un humain.

Dans une base de connaissances décrite en LD, la base de faits (appelée *A-Box* pour *Assertional Box*), contient des faits de la forme $D(a)$ et $R(a,b)$, avec a, b des individus, D un concept et R un rôle. L'ontologie (appelée *T-Box* pour *Terminological Box*) contient des *inclusions de concepts* de la forme $C \sqsubseteq D$ (avec C et D des concepts) et des *inclusions de rôles* de la forme $R \sqsubseteq S$ (avec R et S des rôles) appelés *axiomes*. Ainsi l'axiome

$$Parent \sqcap Humain \sqsubseteq \exists estParentDe.Humain$$

veut dire que tout parent humain est parent d'un humain.

Les axiomes de la T-Box se traduisent en logique du premier ordre. Une inclusion de concepts $C \sqsubseteq D$ sera traduite par la formule logique

$$\forall X\ (\phi_A(X) \rightarrow \phi_B(X))$$

où $\phi_A(X)$ (respectivement, $\phi_B(X)$) est une formule logique dans laquelle X est la seule variable libre traduisant "X est une instance du concept A" (respectivement, B). La construction de cette formule se fait inductivement à partir des traductions données pour chaque constructeur, comme rappelé dans la Table 2.1.

Ainsi l'axiome

$$Parent \sqcap Humain \sqsubseteq \exists estParentDe.Humain$$

se traduit par la formule logique

$$\forall X\ (Parent(X) \wedge Humain(X) \rightarrow \exists Y\ (estParentDe(X,Y) \wedge Humain(Y)))$$

Dès leurs origines, les logiques de description se sont intéressées à la satisfiabilité de la base de connaissances, ou à d'autres problèmes qui peuvent se reformuler en termes de satisfiabilité, ou s'appuyer sur des tests de satisfiabilité, comme la classification de concepts. Par la suite, avec l'essor des données, l'intérêt s'est tourné vers l'interrogation de bases de connaissances avec des requêtes de type requête de bases de données. Cependant, comme les problèmes d'interrogation avec des LDs classiques se sont avérées avoir une grande complexité, d'autres LDs plus simples ont été développées, la plus connue étant la famille DL-Lite [Cal+05]. La Table 2.2 montre en particulier les axiomes autorisés dans la logique DL-Lite$_{\mathcal{R}}$, sans doute le dialecte le plus utilisé de la famille DL-Lite, ainsi que leur traduction en logique du premier ordre.

Syntaxe LD	Traduction logique du 1er ordre
$A, \top, \bot$	$\phi_A(X) = A(X), \phi_\top(X) = true, \phi_\bot(X) = false$
R	$\phi_R(X,Y) = R(X,Y)$
R^-	$\phi_{R^-}(X,Y) = \phi_R(Y,X)$
$C \sqcap D$	$\phi_{C\sqcap D}(X) = \phi_C(X) \wedge \phi_D(X)$
$R \sqcap S$	$\phi_{R\sqcap S}(X,Y) = \phi_R(X,Y) \wedge \phi_S(X,Y)$
$\neg C$	$\phi_{\neg C}(X) = \neg\phi_C(X)$
$C \sqcup D$	$\phi_{C\sqcup D}(X) = \phi_C(X) \vee \phi_D(X)$
$\exists R.C$	$\phi_{\exists R.C}(X) = \exists Y(\phi_R(X,Y) \wedge \phi_C(Y))$
$\forall R.C$	$\phi_{\forall R.C}(X) = \forall Y(\phi_R(X,Y) \rightarrow \phi_C(Y))$

TABLE 2.1 – Principaux constructeurs des LD et leur traduction en logique du premier ordre

2.1.4 Web sémantique

À la différence du web classique, le web sémantique [BLHL01] a pour objectif de partager des connaissances et permettre leur manipulation automatique en qualifiant, formalisant et structurant les données. Il constitue un environnement dans lequel les machines peuvent comprendre le contenu des ressources du web grâce à un système de *métadonnées*, c'est-à-dire d'informations permettant d'en

DL-Lite$_\mathcal{R}$	Traduction logique du 1er ordre
$A \sqsubseteq B$	$\forall X\ (A(X) \rightarrow B(X))$
$A \sqsubseteq \exists R.\top$	$\forall X(A(X) \rightarrow \exists Y\ R(X,Y))$
$R \sqsubseteq \exists S^-$	$\forall X \forall Y\ (R(X,Y) \rightarrow \exists Z\ S(Z,X))$
$B \sqsubseteq \exists R.C$	$\forall X\ (B(X) \rightarrow \exists Y\ (R(X,Y) \wedge C(Y)))$
$R \sqsubseteq S$	$\forall X \forall Y\ (R(X,Y) \rightarrow S(X,Y))$
$B \sqsubseteq \neg C$	$\forall X\ (B(X) \wedge C(X) \rightarrow \bot)$

TABLE 2.2 – Axiomes de DL-lite$_\mathcal{R}$ et leur traduction en logique du premier ordre

décrire d'autres. Les technologies du web sémantique [HKR09] reposent principalement sur la famille de langages développés par le *W3C*, World Wide Web Consortium [2].

Parmi les principaux langages formels du web sémantique, on trouve le langage d'annotation *RDF*, pour Resource Description Framework [3], son extension *RDFS* (RDF Scheme) [4] ajoutant des connaissances ontologiques de base, ainsi que la famille de langages OWL, pour Web Ontology Language [5] consacré à la description d'ontologies plus ou moins complexes.

Le langage RDF permet de décrire formellement des ressources web et leurs métadonnées. Ce formalisme est le langage de base du web sémantique car il fournit de façon concrète l'interopérabilité entre les applications qui échangent des informations sur le web. Il permet de formaliser les métadonnées dans les documents et ainsi de les rendre accessibles à des traitements automatisés.

Un graphe RDF est un ensemble de triplets de la forme *(sujet, prédicat, objet)* où le sujet est une ressource désignée par un identifiant (IRI - donnant l'adresse web de la ressource - ou Blank désignant une ressource anonyme), le prédicat (ou propriété) définit la relation binaire entre le sujet et l'objet, et l'objet est une ressource désignée par un identifiant (IRI, Blank) ou bien une valeur littérale. En particulier, la propriété `rdf:type` permet de typer les objets en leur associant une classe. On représente un ensemble de triplets sous la forme d'un graphe étiqueté où les sommets sont les sujets et objets des triplets et chaque arc correspond à un triplet ; un sommet est étiqueté par le sujet ou l'objet associé ; un arc correspondant à un triplet *(s p o)* va du sommet associé à s vers le sommet associé à o et est étiqueté par p (voir la figure 2.2 qui dit qu'Alice est la mère de Bob, qui est une personne).

RDF Schema (RDFS) est une extension de RDF qui ajoute des notions per-

2. https://www.w3.org
3. https://www.w3.org/RDF/
4. https://www.w3.org/TR/rdf-schema/
5. https://www.w3.org/OWL/

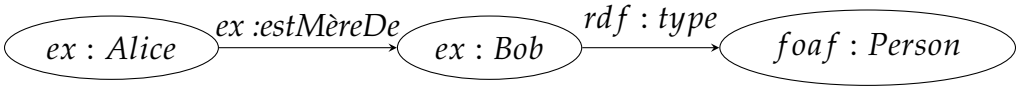


FIGURE 2.2 – Exemple de graphe RDF

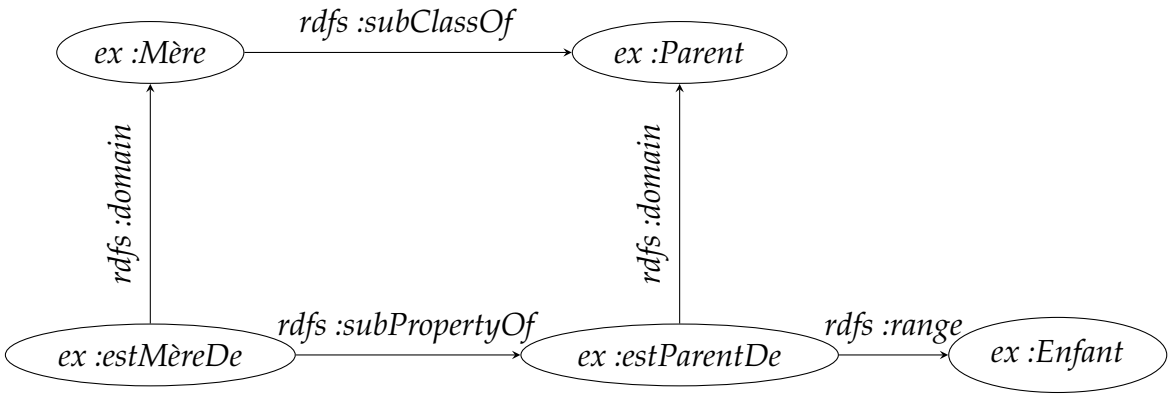


FIGURE 2.3 – Exemple de graphe RDFS

mettant la représentation d'ontologies simples. Plus précisément, une ontologie RDFS déclare des contraintes sémantiques entre les classes et les propriétés à l'aide des propriétés prédéfinies suivantes :

- `rdfs:subClassOf` pour définir des hiérarchies de classes.
- `rdfs: subPropertyOf` pour définir des hiérarchies de relations.
- `rdfs: domain` pour définir la classe des sujets d'une propriété (le domaine de la propriété).
- `rdfs: range` pour définir la classe (ou le type de données) des objets d'une propriété (le co-domaine de la propriété).

La figure 2.3 montre le graphe de cinq triplets affirmant que *Mère* est une spécialisation (sous-classe) de *Parent*, *estMèreDe* est une spécialisation (sous-propriété) de *estParentDe*, *estParentDe* a pour domaine *Parent* et co-domaine *Enfant*, *estMèreDe* a pour domaine *Mère*.

Il existe plusieurs façons de traduire des triplets RDF(S) en logique. Nous considérons ici l'approche classique de la représentation de connaissances qui sépare strictement les classes, des propriétés et des instances. Ainsi, une classe se traduit en un prédicat unaire, une propriété en un prédicat binaire, et une instance en un terme qui est soit une constante (IRI, littéral), soit une variable (Blank). Cette traduction logique ne permet pas de traduire n'importe quel ensemble de triplets RDF(S) mais seulement ceux qui respectent la séparation entre classes, propriétés et instances.

$$\forall X\ (\mathit{Mère}(X) \rightarrow \mathit{Parent}(X))$$
$$\forall X \forall Y\ (\mathit{estMèreDe}(X,Y) \rightarrow \mathit{estParentDe}(X,Y))$$
$$\forall X \forall Y\ (\mathit{estParentDe}(X,Y) \rightarrow \mathit{Parent}(X))$$
$$\forall X \forall Y\ (\mathit{estParentDe}(X,Y) \rightarrow \mathit{Enfant}(Y))$$
$$\forall X \forall Y\ (\mathit{estMèreDe}(X,Y) \rightarrow \mathit{Mère}(X))$$

FIGURE 2.4 – Traduction en logique de la Figure 2.3

Ainsi, un graphe RDF donne lieu à une conjonction d'atomes qui peuvent prendre deux formes :

- $C(s)$ pour un triplet de la forme *(s rdf :type C)*, où on désigne ici par s le sujet du triplet et le terme associé, et par C l'objet du triplet et le prédicat unaire associé ;
- $p(o,s)$ pour un triplet de la forme $(s\ p\ o)$, où $p \neq \mathit{rdf:type}$, et on identifie à nouveau le sujet et l'objet du triplet avec les termes associés.

Si le graphe RDF ne comporte pas de Blank, on obtient ainsi une base de faits à notre sens (sinon, on obtient une notion de base de faits plus générale, où on peut avoir des variables, qui seront quantifiées existentiellement, et représenteront des entités ou valeurs inconnues). Par exemple, le graphe RDF de la Figure 2.2 se traduit en logique par l'ensemble de faits $\{\mathit{estMèreDe}(\mathit{alice},\mathit{bob}), \mathit{personne}(\mathit{bob})\}$ (où *alice* correspond à $ex : Alice$, *bob* à $ex : Bob$ et *person* à $foaf : Person$).

Les triplets RDFS proprement dits se traduisent en des règles, comme illustré dans la figure 2.4, qui donne la traduction en logique des triplets de la Figure 2.3.

Le langage *OWL*, basé sur RDF et RDFS, permet d'étendre les possibilités de RDFS et de décrire ainsi des ontologies plus riches. OWL se compose de 3 sous-langages : *OWL lite*, *OWL DL* et *OWL Full*. OWL lite est le sous-langage le plus simple et est destiné à représenter des hiérarchies de concepts. OWL DL est plus expressif que OWL Lite et est fondé sur les logiques de description. Il est adapté pour effectuer des raisonnements complets (c'est-à-dire que toutes les déductions logiques possibles sont réalisées) et décidables (c'est-à-dire toutes les déductions peuvent s'effectuer en temps fini). Le sous-langage le plus expressif est OWL Full, qui ne garantit pas forcément la complétude et la décidabilité des calculs.

La dernière version de OWL (OWL2) [6] comporte des sous-langages basés sur des LD légères, en particulier OWL2 QL qui vise à répondre efficacement à des requêtes conjonctives sur de grandes bases de faits (ou de données). On notera que OW2 QL est basé sur la LD DL-Lite$_{\mathcal{R}}$, que nous avons décrite précédemment.

6. https://www.w3.org/TR/owl2-overview/

2.1.5 Langages à base de règles

De façon générale, une règle est une connaissance de la forme "si [condition] alors [conclusion]" avec la sémantique intuitive suivante "si la partie condition est vérifiée par la base de faits, alors la partie conclusion produit un (ou des) nouveau(x) fait(s)".

Il existe de nombreux langages à base de règles plus ou moins expressifs. Nous commençons par présenter le langage *Datalog* [CGT+89 ; AHV94], sur lequel nous nous sommes appuyés dans cette thèse. Datalog est originellement un langage de requêtes pour les bases de données, plus expressif que les requêtes du premier ordre, auxquelles il ajoute la récursivité ; mais par la suite l'expression "règles Datalog" a été utilisée pour décrire des connaissances. C'est dans ce sens que nous utilisons le terme Datalog.

Le noyau de Datalog est composé de règles positives dans lesquelles toutes les variables de la partie conclusion de la règle apparaissent dans la partie condition. Ces règles prennent la forme générale suivante :

$$\forall \overrightarrow{X} \forall \overrightarrow{Y} \ (\ Corps[\overrightarrow{X}, \overrightarrow{Y}] \rightarrow Tête[\overrightarrow{X}])$$

où le *corps* (ou condition) et la *tête* (ou conclusion) sont des conjonctions d'atomes, $\overrightarrow{X}$ et $\overrightarrow{Y}$ deux tuples de variables. Notons que, contrairement à la définition usuelle des règles Datalog, nous considérons ici que la tête peut contenir plusieurs atomes. Ceci ne change rien à l'expressivité du langage (la règle $C \rightarrow t_1 \wedge \ldots \wedge t_k$ est équivalente à la conjonction des k règles $C \rightarrow t_i$), mais améliore à la fois la concision de l'écriture et le temps de calcul (on ne teste qu'une seule fois l'applicabilité de la règle de corps C au lieu de k fois).

Les règles Datalog peuvent notamment exprimer des spécialisations de concept ou de relations, ou définir des signatures de relations.[7] Par exemple, les règles de la figure 2.4 sont des règles Datalog. Il est immédiat de voir que Datalog généralise strictement RDFS. D'une part, l'arité des prédicats n'est pas forcément unaire ou binaire, d'autre part les corps de règles ont une structure quelconque.

Datalog a par la suite été étendu pour prendre en compte d'autres formes de règles ou des règles plus expressives, notamment :

- des *contraintes négatives*, qui sont de la forme

$$\forall \overrightarrow{X} \ (Corps(\overrightarrow{X}) \quad \rightarrow \quad \bot)$$

où le *corps* est également une conjonction d'atomes et le symbole $\bot$ ("ab-

7. Une signature d'une relation fixe le type de chaque argument de cette relation.

surde") est toujours faux. A noter qu'une contrainte négative est équivalente à la formule $\neg \exists \overrightarrow{X} Corps(\overrightarrow{X})$. Par exemple,

$$\forall X \ (Père(X) \wedge Mère(X) \rightarrow \bot)$$

signifiant que "une personne ne peut pas être à la fois un père et une mère".

- des règles avec négation (dans le corps de règle), de la forme

$$\forall \overrightarrow{X} \forall \overrightarrow{Y} \ (Corps^{+}[\overrightarrow{X}, \overrightarrow{Y}] \wedge Corps^{-}[\overrightarrow{X}, \overrightarrow{Y}] \rightarrow \ Tête[\overrightarrow{X}])$$

où le *corps positif* $corps^{+}$ est une conjonction d'atomes, le *corps négatif* $corps^{-}$ est une conjonction de négations d'atomes, et la *tête* définit également une conjonction d'atomes.
Par exemple,

$$\forall X \forall Y \ (estParentDe(X, Y) \wedge \neg estPèreDe(X, Y) \rightarrow estMèreDe(X, Y))$$

signifiant que "si X est un parent de Y mais n'est pas son père, alors X est la mère de Y".

Il existe différentes manières de gérer la négation dans les règles suivant les hypothèses sémantiques adoptées. En Datalog, on considère que la négation est la "négation par défaut", exprimant l'absence d'un fait. Plus de détails seront donnés au chapitre 3.

Notons par ailleurs que Datalog et les logiques de description classiques sont incomparables en expressivité. D'un côté Datalog permet d'exprimer des relations complexes sur des variables dans le corps des règles, alors que les logiques de description ne peuvent exprimer que des relations "arborescentes". Par exemple, la connaissance "si X est un parent de Y et de Z alors Y et Z sont frères et soeurs" est facilement exprimable en Datalog comme suit

$$\forall X \forall Y \forall Z \ (estParentDe(X, Y) \wedge estParentDe(X, Z) \rightarrow sontFrèresEtSoeurs(Y, Z))$$

mais difficilement exprimable en logique de description (on aurait besoin d'un constructeur complexe, la composition de rôles, pour pouvoir parler du lien entre Y et Z). Des relations plus complexes entre les variables ne peuvent pas du tout s'exprimer en LD [Bor96].

D'un autre côté, les logiques de description permettent de conclure à l'existence d'entités qui n'existent pas forcément dans la base de faits. Par exemple, "tout humain a un parent" est facilement exprimable en LD par l'axiome $Humain \sqsubseteq$

$\exists aPourParent$ (ou bien : $Humain \sqsubseteq \exists estParentDe^{-}$) mais n'est pas exprimable en Datalog.

Finalement, mentionnons la famille des règles existentielles (voir [Mug11; Got+12; MT14] pour des introductions à ce formalisme) qui généralisent à la fois Datalog et les logiques de description utilisées pour l'accès aux données (la famille DL-Lite notamment). Ces règles sont de la forme :

$$\forall \overrightarrow{X} \forall \overrightarrow{Y} \ (Corps[\overrightarrow{X}, \overrightarrow{Y}] \rightarrow \exists \overrightarrow{Z} \ Tête[\overrightarrow{Y}, \overrightarrow{Z}])$$

où le corps et la tête sont également des conjonctions finies d'atomes.

Comme nous le voyons dans la forme générale ci-dessus, ces règles contiennent en tête des variables ($\overrightarrow{Z}$) qui n'apparaissent pas dans le corps et qui sont quantifiées existentiellement. Ceci permet d'inventer de nouveaux objets ou plus précisément d'affirmer l'existence d'objets dont on ne sait pas s'ils sont égaux à des objets déjà connus ou pas. Par exemple, si on reprend l'exemple de l'axiome "Tout humain a un parent", celui-ci est exprimable avec une règle existentielle comme suit :

$$\forall X \ (Humain(X) \rightarrow \exists Y \ estParentDe(Y, X))$$

2.1.6 Accès aux données

Dans cette section, nous nous intéressons à l'accès à des données issues de différentes sources. Cet accès passe par le processus d'*intégration de données* [DHI12; Abi+11], qui consiste à fournir un accès unifié à des données provenant de multiples sources de données, définissant ainsi une base de données globale (Figure 2.5). La base de données globale peut être effectivement construite ou rester virtuelle.

L'étape qui consiste à sélectionner une partie des données d'une source et de la traduire en des données sur la base globale est faite à l'aide de *mappings* (représentés par des flèches noires sur la Figure 2.5).

De manière générale, un mapping peut être vu comme une règle allant d'une structure [8] S_1 vers une structure S_2 de la forme :

$$q_{S_1}(\overrightarrow{X}) \rightarrow q_{S_2}(\overrightarrow{X})$$

tel que q_{S_1} est une requête d'interrogation exprimée sur la structure S_1, dont les variables réponses sont $\overrightarrow{X}$, et q_{S_2} une requête d'insertion dans la structure S_2,

8. Une structure peut être une base de données relationnelle ou autre type de base de données, un fichier structuré de type *csv*, une base de faits, etc

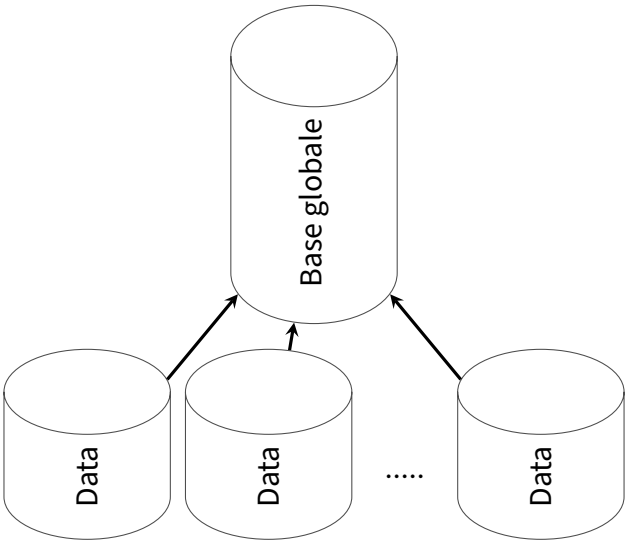


FIGURE 2.5 – Intégration de données. Les flèches représentent des mappings.

qui insère les réponses retournées par q_{S_1}.

Dans ce cadre d'intégration de données, les requêtes sont formulées en utilisant le schéma de la base de données globale. Deux grandes approches pour l'évaluation des requêtes sont possibles :

- soit l'approche par *matérialisation* (aussi appelée approche *ETL*, pour *Extract, Transform, Load*) qui consiste à déclencher les mappings pour construire effectivement la base de données globale et les requêtes sont évaluées sur cette base globale ;
- soit l'approche par *virtualisation*, qui consiste à garder la base globale virtuelle, et toute requête q_G sur la base globale est réécrite avec les mappings en une union de requêtes Q_s sur les sources, de façon à ce que l'évaluation de Q_s sur les sources admette le même ensemble de réponses que q_G sur la base globale si elle était matérialisée.

Le cadre *Ontology-Based Data Access (OBDA)* [Pog+08 ; Xia+18] est un nouveau paradigme, qui marie l'intégration de données (Figure 2.5) et le raisonnement sur des bases de connaissances (Figure 2.1). Au lieu d'avoir un simple schéma global, on a une ontologie (autrement dit, au lieu d'avoir une base de données globale, on a une base de connaissances). Cette ontologie fournit non seulement un vocabulaire commun, mais aussi la capacité de faire des inférences. Comme nous pouvons le voir dans la Figure 2.6, l'architecture d'un système OBDA comporte trois niveaux : le *niveau conceptuel* qui est défini par l'ontologie ; le *niveau des données*, pouvant intégrer des sources de données indépendantes et hétérogènes ; et le *niveau des mappings*, qui fait la correspondance entre le niveau des données, en

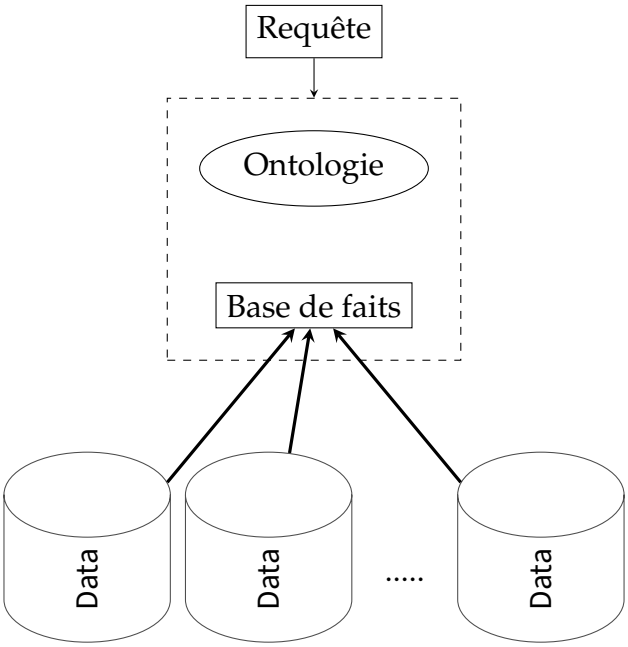


FIGURE 2.6 – Architecture OBDA

sélectionnant celles qui sont pertinentes, et le niveau conceptuel en définissant la base de faits qui utilise le vocabulaire de l'ontologie. Les requêtes à un système OBDA sont formulées au niveau conceptuel, c'est-à-dire en utilisant le vocabulaire de l'ontologie.

Puisqu'en OBDA on remplace le schéma de la base de données globale par une ontologie, il faut prendre en compte l'ontologie dans le mécanisme de réponse aux requêtes :

- soit la base de faits est matérialisée en activant les mappings et elle est enrichie avec tous les faits qui peuvent être inférés à partir de l'ontologie (saturation) ; ensuite une requête q_O est évaluée sur la base de faits saturée ;
- soit la base de faits n'est pas matérialisée et une requête q_O est d'abord réécrite avec l'ontologie puis avec les mappings en une union de requêtes Q_s, de façon à ce que l'évaluation de Q_s sur les sources fournisse le même ensemble de réponses que l'évaluation de q_O sur la base de faits saturée.

À noter que, dans le cadre de la matérialisation, on peut considérer une approche intermédiaire qui ne sature pas la base de faits : dans ce cas, la requête est réécrite avec l'ontologie, puis évaluée sur la base de faits (matérialisée mais non saturée).

Les systèmes OBDA existants se placent dans le cadre du web sémantique. Ils sont basés sur des langages de représentation de connaissances peu expressifs, comme OWL 2 QL (le dialecte de OWL 2 basé sur DL-Lite) ou RDFS. Les deux

systèmes les plus matures, *Ontop* [9] [Cal+15] et *MASTRO* [10] [Cal+11] sont associés à OWL 2 QL et mettent en oeuvre une approche de virtualisation. Il faut préciser par ailleurs que la saturation avec DL-Lite (OWL 2 QL) ne se termine pas toujours, mais que la réécriture des requêtes considérées est toujours finie. Ontop représente le contenu d'une base de données relationnelle sous forme d'un graphe RDF virtuel à l'aide de mappings R2RML (langage pour exprimer des mappings de bases de données relationnelles vers des ensembles de triplets RDF). Les requêtes sont écrites en SPARQL [11], le langage d'interrogation de graphes RDF, et sont reformulées (avec l'ontologie) puis récrites (avec les mappings) en des requêtes SQL exécutées par le système de gestion de la base relationnelle. MASTRO adopte les mêmes principes que Ontop.

Citons aussi *Ultrawrap* [SM13] basé sur une extension de RDFS et *Obi-wan* [Bur+20a] basé sur RDFS. À la différence des systèmes précédents qui considèrent une seule base de données relationnelle (éventuellement obtenue en intégrant plusieurs bases de données), Obi-Wan gère directement un ensemble de bases de données aux modèles hétérogènes, via un médiateur capable d'interroger ces différentes sources. Il n'est pas capable de traiter tout SPARQL, mais seulement le noyau de SPARQL correspondant approximativement aux requêtes conjonctives ; par contre, il permet de formuler des requêtes SPARQL portant à la fois sur les triplets RDFS de l'ontologie et les triplets RDF provenant des données, alors que les systèmes précédents ne permettent d'interroger que les triplets RDF associés aux données. Le système Ultrawrap combine matérialisation et virtualisation. Obi-wan offre différentes stratégies allant de la matérialisation à la virtualisation.

Ajoutons que des travaux sont en cours pour construire un système OBDA basé sur le langage expressif des règles existentielles. Ces travaux étendent Graal, un moteur de raisonnement sur les règles existentielles [12] [Bag+15].

Dans le chapitre suivant, nous présenterons notre cadre formel que l'on peut brièvement exprimer ainsi : nous considérons une architecture de type OBDA, avec une approche de matérialisation et un langage de représentation de connaissances basé sur Datalog. Plus précisément, ces règles correspondent à une extension du Datalog positif avec des prédicats et fonctions calculés et une forme de négation (appelée stratifiée).

Plusieurs raisons justifient le choix de la matérialisation dans notre cas d'étude : d'abord, la virtualisation a été principalement développée pour des requêtes simples

9. https://ontop-vkg.org
10. http://obdm.obdasystems.com/mastro/
11. https://www.w3.org/TR/rdf-sparql-query/
12. https://gitlab.inria.fr/rules/graal-v2

(essentiellement des unions de requêtes conjonctives), alors que les requêtes jugées pertinentes dans notre cas sont plus complexes : elles peuvent notamment impliquer des agrégations. Ensuite, certaines caractéristiques de notre langage de représentation des connaissances (fonctions calculées, négation par défaut) ne permettent pas d'utiliser des techniques de reformulation de requêtes. De plus, la plupart des requêtes utilisateurs nécessitent de classer les espèces (par exemple : "Trouver les k meilleures espèces pour un service donné") et la matérialisation est plus adaptée pour répondre efficacement à de telles requêtes. Enfin, le principal avantage de la virtualisation est l'indépendance par rapport à l'évolution des sources de données, mais cela ne semble pas être un problème dans notre cas.

2.2 État de l'art en ingénierie agroécologique

La transition agroécologique nécessite de mettre en place de nouveaux types de systèmes de cultures répondant à la fois à des critères économiques, environnementaux et sociaux. Ces systèmes étant complexes, leur conception ne suit pas une méthode prédéfinie à appliquer. Actuellement, elle passe notamment par l'organisation d' "ateliers de conception" réunissant des experts de différents domaines (agronomie, écologie, etc). Ces ateliers mobilisent des connaissances scientifiques, mais également des savoir-faire agricoles, des retours d'expériences, des informations pertinentes issues de sources de diverses données, etc. L'activité de conception se heurte à deux difficultés : (i) les différents types d'informations mobilisés sont de nature très hétérogène et se situent dans des sources dispersées ; et (ii) les connaissances scientifiques en agroécologie sont en pleine évolution et ne sont pas stabilisées, donc susceptibles d'évoluer rapidement.

L'organisation d'ateliers est très coûteuse en temps et l'activité de conception encore peu aidée par des outils informatiques. Il y a donc un intérêt crucial à développer "l'ingénierie agroécologique" (agroecological engineering, [Les+15a]), ce qui passe par la définition et l'implémentation de cadres informatiques "intelligents" permettant un accès unifié à des sources de données hétérogènes et fournissant des techniques permettant de les comparer, analyser, combiner, de façon à aider à la création de nouvelles connaissances utiles à la recherche et à l'innovation.

2.2.1 Approches sémantiques en ingénierie agroécologique

Les approches sémantiques prennent une importance croissante dans l'harmonisation et la formalisation des données. Ainsi, de nombreuses initiatives dans le secteur de l'agronomie ont permis l'éclosion de projets qui utilisent les technologies du *web sémantique* afin de développer des ressources sous la forme de thésaurus et d'ontologies, et d'outils exploitant ces ressources.

Un thésaurus peut être défini comme un ensemble de termes contrôlés en langue naturelle, généralement dans plusieurs langues, avec leurs liens sémantiques, qui sont parfois mis en relation avec des concepts. Comme on l'a vu précédemment, les ontologies sont une notion plus formalisée, au sens où elles s'appuient sur un langage mathématique (logique) qui leur donnent une sémantique précise et non ambigüe. Les thésaurus servent essentiellement à indexer des contenus (textes, images, etc) et à les rechercher, alors que les ontologies permettent d'effectuer des raisonnements.

Parmi les ressources et outils participant à une approche sémantique de l'ingénierie agroécologique, nous pouvons citer notamment :

- Des *thésaurus* comme *TOP* (Thesaurus Of Plant characteristics) [Gar+17], un thésaurus sur les caractéristiques des plantes qui fournit des noms, des définitions, des unités formelles et des synonymes pour plus de 700 traits végétaux ; ou *AGROVOC* [RK12] qui offre un vocabulaire contenant plus de 40 100 concepts relatifs à l'agriculture, liés à 939 000 termes en près de 41 langues.

- Des *ontologies* comme *Plant Ontology* [Jai+05], qui fournit à la fois un vocabulaire structuré et une base de données qui relie l'anatomie, la morphologie, la croissance et le développement des plantes aux données de la génomique végétale ; *Crop Ontology* [Mat+13], qui fournit des descriptions des caractéristiques agronomiques, morphologiques, physiologiques, de qualité et de stress, ainsi que leurs définitions et relations.

- Des *portails*, qui fournissent des thésaurus et des ontologies, comme *AgroPortal* [Jon+18], qui est un référentiel de thésaurus et d'ontologies pour l'agronomie et les domaines connexes. AgroPortal inclut 145 ontologies (comprenant PO) et 12 thésaurus (incluant TOP). Les thésaurus sont décrits majoritairement en *SKOS* (pour Simple Knowledge Organization System) [MB09] et les ontologies, majoritairement en OWL (113) avec des

classes (647 521 au total), des propriétés et des instances de ces classes. En pratique les classes sont organisées en une hiérarchie (lien classe / sous-classe) mais la définition des classes va rarement au delà, ne permettant donc que des raisonnements assez limités. Par contre, des fonctionnalités importantes fournies par de tels portails sont celles liées aux alignements entre les différentes ontologies.

- Des *outils de capitalisation des connaissances* utilisant des ontologies afin d'améliorer le partage des connaissances en agroécologie comme
  (i) *GECO* [Sou+19], qui est un outil collaboratif de gestion de connaissances et d'échanges, dédié à la transition agroécologique. L'outil GECO a été développé par l'INRAe, l'ACTA et le RMT (Réseau Mixte Technologique) "Systèmes de cultures innovants". L'objectif de cet outil est de permettre la consultation, la co-construction et l'échange des informations. GECO constitue un moteur de recherche et permet l'accès à des informations disponibles sous forme de fiches (comportant des textes et des figures) et sous forme de discussions thématiques. Les informations présentes dans ces 2 espaces sont décrites par des méta-données qui utilisent les concepts d'une ontologie. L'accès à ces différents contenus se fait donc à partir des méta-données ;
  (ii) le prototype *SYGNAL* [Léo+20] qui permet l'acquisition d'un modèle conceptuel d'agrosystème sous forme d'un graphe connectant des concepts (notamment *Agent*, concept qui permet de spécifier une entité physiquement observable à un instant t et *Processus*, qui permet de figurer les évènements mettant en action et en interaction les agents) par des relations binaires (comme *implique* qui va d'un Processus à un Agent et permettant de lier les agents aux processus dans lesquels ils sont impliqués). Des requêtes simples (graphes correspondant à des requêtes conjonctives) peuvent être effectuées avec l'éditeur de graphes conceptuels Cogui et des requêtes plus complexes par programmation en Python. Il faut noter cependant que les graphes obtenus n'ont pas de traduction formelle complète dans un langage de représentation de connaissances et de raisonnement existant.

- Des *logiciels open-source* permettant de construire des applications comme *OpenSilex* [13]. OpenSilex est un ensemble de modules logiciels interopérables utilisés pour la création de systèmes d'information scientifiques dédiés à l'exploitation de données à grandes échelles en agronomie. L'ar-

13. http://www.opensilex.org/

chitecture proposée dans OpenSilex est ressemblante à celle d'un système OBDA. En effet, comme pour OBDA, il y'a une distinction en 3 niveaux : le niveau des *données* (comportant des données de différents formats), le niveau des *connaissances* (comprenant les triplets RDF) et des *services web*, faisant le lien entre les 2 niveaux précédents et qui permettent d'importer des données de différents formats au format RDF. Le composant RDF4J permet d'interroger des données RDF (un triple store), en SPARQL notamment.

Les ressources précédentes permettent de capitaliser les connaissances et de les rendre plus faciles d'accès, d'utilisation et de partage. Ceci passe par la mise en place de cadres plus ou moins formels rassemblant les notions essentielles du domaine agricole dans un vocabulaire bien défini sous la forme de thésaurus ou d'ontologies. Néanmoins, dans leur état actuel, ces ressources ne permettent que très peu de raisonnements sur les connaissances. Dans notre cas d'étude, nous aurons besoin de représentations plus complexes, qui mettent en oeuvre des règles (extensions de Datalog) provenant des connaissances expertes, ce qui ne peut se faire en RDF, et généralement pas en OWL non plus.

2.2.2 Outils d'aide à la sélection d'espèces

Ayant pour objectif principal de concevoir un outil proposant des espèces associées à certains services écosystémiques, nous nous intéressons maintenant aux outils existants ayant un objectif similaire. S'il existe de nombreux outils d'aide à la sélection d'espèces pour les agriculteurs, peu d'entre eux considèrent la notion de service écosystémique. Parmi ces derniers, nous pouvons citer notamment :

• **L' outil *CAPS*** [14] (Colza Associé à des Plantes de Service), qui a été développé dans le cadre du projet CASDAR Alliance [15], a pour objectif, entre autres, de quantifier les services écosystémiques rendus par l'association de plantes de services pour le colza d'hiver. Cet outil, qui s'adresse aux agriculteurs, conseillers, étudiants et enseignants du secteur agricole, permet d'aider les utilisateurs à identifier les espèces de plantes les plus performantes pour les services voulus dans un contexte donné. L'outil couvre 3 services écosystémiques : *la fourniture d'azote à la plante cultivée*, *la perturbation des insectes ravageurs à l'automne* et *la limitation des adventices à l'automne* et 11 espèces (dont 9 légumineuses) et adopte une

14. https://www6.versailles-grignon.inra.fr/agronomie/Recherche/Regulations-biologiques/Projet-CASDAR-Alliance/Caps-Colza-associe
15. https://www6.versailles-grignon.inrae.fr/agronomie/Recherche/Axe-2-Les-biodiversites-et-les-services-ecosystemiques-rendus/Projets-de-recherche/Projet-CASDAR-Alliance

approche basée sur les traits fonctionnels permettant de quantifier le rendu des services par ces espèces. En effet, dans le cadre de ce projet, une collecte de données expérimentale a été faite pour produire une base de données comprenant des valeurs de 24 traits pour ces 11 espèces considérées. Les traits considérés ici ont des valeurs qualitatives (faible, moyen, etc).
En plus de cette base de données créée, des tableaux reliants les traits aux services ont été construits à travers des ateliers participatifs regroupant des experts (chercheurs, semenciers, ingénieurs d'institut techniques, conseillers de chambres d'agriculture) de domaines différents et à l'aide de la bibliographie, permettant le passage des traits aux fonctions puis au service par une agrégation des valeurs. L'agrégation des informations s'est faite grâce au logiciel d'aide à la décision Dexi, basé sur l'approche multicritère [Boc14]. Ce logiciel permet de développer des modèles se composant de critères (qui sont des variables qualitatives) ; d'échelles de valeurs (pouvant être prises par les critères) ; d'un arbre de critères et de règles de décision, comme par exemple : *si <critère1 est très faible> et si <critère2 est moyen> alors <le critère agrégé est faible>* qui définissent les agrégations des critères depuis les feuilles jusqu'à la racine de l'arbre. Ces règles sont fixées par les experts et prennent en compte des valeurs qualitatives de traits ainsi que des informations de contexte. Notons que DEXi ne gère pas d'autres types de règles, en effet, les règles définies dans un modèle décisionnel construit sur ce logiciel ne permettent que d'agréger des critères.

• **L'outil *SIMSERV*** [OL+10], qui a été mis en place par l'INRAe dans l'objectif d'optimiser la sélection d'espèces susceptibles de fournir un ou plusieurs services en réponse à un contexte pédoclimatique et socio-économique donné. La démarche adoptée pour cet outil est très ressemblante à celle du précédent. En effet, à l'aide de connaissances expertes et de la bibliographie, une base de données a été construite dans laquelle sont stockées les données concernant les plantes de service (environ une centaine de plantes de service y sont répertoriées), les parcelles (intégrant les contextes pédoclimatique et socio-économique) et deux cultures de rente : la banane et l'igname. Par la suite, en utilisant également le logiciel DEXi, et à l'aide d'un arbre de décision et de règles de décision construites dans ce logiciel, le rendu des espèces est évalué pour les services considérés et comme précédemment, une approche basée sur des traits fonctionnels. Au total, cinq services sont pris en compte dans cet outil : *la lutte contre l'érosion*, *la lutte contre les adventices*, *la régulation des bioagresseurs* notamment *la nématorégulation*, *l'apport des éléments nutritifs*, et *le tuteurage*.

• **Un outil d'agroforesterie** [VDW+19], qui est un outil d'aide à la décision et

à la sélection d'arbres d'ombrage à implémenter dans un système de culture en tenant compte des conditions locales (Ouganda, Ghana) et des préférences des exploitants agricoles tout en maximisant les services écosystémiques de la parcelle. Cet outil est spécifiquement prévu pour les cultures de café et de cacao dans les conditions de l'Ouganda et du Ghana. Les données utilisées dans cet outil ont été collectés par des interviews faites auprès d'exploitants agricoles. En effet, après avoir identifié une région, cerné les services écosystémiques et les espèces d'arbres d'ombrage à étudier, il a été demandé aux exploitants agricoles d'ordonner les différentes espèces sélectionnées pour les services voulus. Une fois cette étape de collecte de données réalisée, leur analyse s'est faite via le package BradleyTerry2 [TF12] du langage R afin d'obtenir un score pour chacune des espèces d'arbres classées, et donc un ordre total entre les espèces, calculé en fonction des différents classements faits par les exploitants. L'analyse de Bradley Terry fournit également un intervalle de confiance, qui peut être considéré comme un indicateur de la fréquence et de l'homogénéité avec lesquelles les agriculteurs classent les espèces d'arbres. Des intervalles de confiance plus grands sont associés à des espèces d'arbres que les agriculteurs ont tendance à sélectionner moins fréquemment et à classer de manière moins concordante. Comme les intervalles de confiance peuvent varier de manière assez significative entre les arbres, l'ordonnancement final des arbres proposé à l'utilisateur ne se fait pas uniquement sur l'ordre des scores, mais sur une combinaison incluant à la fois le score et la taille de l'intervalle de confiance. Pour finir, la mise au point de l'outil et la validation des résultats, s'est effectuée par des révisions par les pairs (discussions de groupe, interviews des agriculteurs, révisions par des experts).

Bien que ces outils fournissent à l'utilisateur des résultats précis sur le rendu des services par des espèces dans un contexte déterminé par ce dernier, nous remarquons qu'ils sont très spécifiques à un cas particulier. En effet, l'outil CAPS est conçu spécifiquement pour le colza et 11 espèces de service associées ; l'outil SIMSERV couvre les cultures de banane et d'igname uniquement en fixant une liste d'espèces déjà identifiées ; et l'outil en agroforesterie prend en compte les cultures de café et de cacao dans un contexte spécifique donné et pour des espèces déjà choisies. Bien que ces outils donnent des résultats "pointus" et fiables (issus de retours d'expérience), ils présentent un certain nombre de limitations. D'une part, ils manquent de *généricité*, au sens où leur adaptation à d'autres systèmes de culture, dans d'autres conditions de contexte, nécessitent de revoir toute la modélisation ; d'autre part, ils sont peu *évolutifs*, au sens où une évolution des

connaissances en agroécologie (par exemple sur les relations entre les traits fonctionnels et les services) nécessite également de revoir toute la modélisation, alors même que les connaissances de ce domaine ne sont pas stabilisées, donc susceptibles d'évoluer rapidement. De plus, la liste des espèces étant déjà fixée, ces outils ne peuvent pas proposer de *nouvelles espèces*, jamais encore testées, à intégrer dans le système, alors même que des données sur ces espèces sont disponibles à travers certaines sources de données. Enfin, aucun des outils ne propose un mécanisme *d'explication* des résultats à l'utilisateur. On peut remarquer que les outils CAPS et SIMSERV reposent sur des règles de décision (écrites dans DEXi) qui pourraient potentiellement servir de base à un mécanisme d'explication dès lors que les critères de sélection ont du sens pour l'utilisateur.

Notre objectif est de proposer un cadre permettant de construire un outil :

- suffisamment *générique* pour être applicable à différents cas d'étude visant la sélection de plantes de services basée sur une approche traits - fonctions - service ;
- *capable d'évoluer facilement*, que ce soit au niveau des données ou des connaissances ;
- *sans limitation* a priori *des espèces* de service considérées, afin d'ouvrir le champ des possibles et de pouvoir proposer à l'utilisateur de nouvelles espèces candidates au rendu d'un service qu'il pourra par la suite expérimenter ;
- dans lequel les *raisonnements* mis en oeuvre peuvent être rendus compréhensibles par un utilisateur.

Comme nous l'expliquerons de façon détaillée dans le chapitre suivant, nous avons fait le choix de la conception d'un outil basé sur une architecture OBDA, qui combine des connaissances expertes sur les liens entre traits fonctionnels et services écosystémiques avec des bases de données sur les traits, d'une façon générique (c'est-à-dire non liée à un cas d'étude particulier) et explicable (c'est-à-dire permettant de justifier les résultats auprès d'un utilisateur).

CHAPITRE 3

Cadre formel

Dans ce chapitre, nous présentons notre cadre formel : l'architecture générale inspirée de OBDA, le langage de représentation de connaissances basé sur Datalog, et les différents types de mappings.

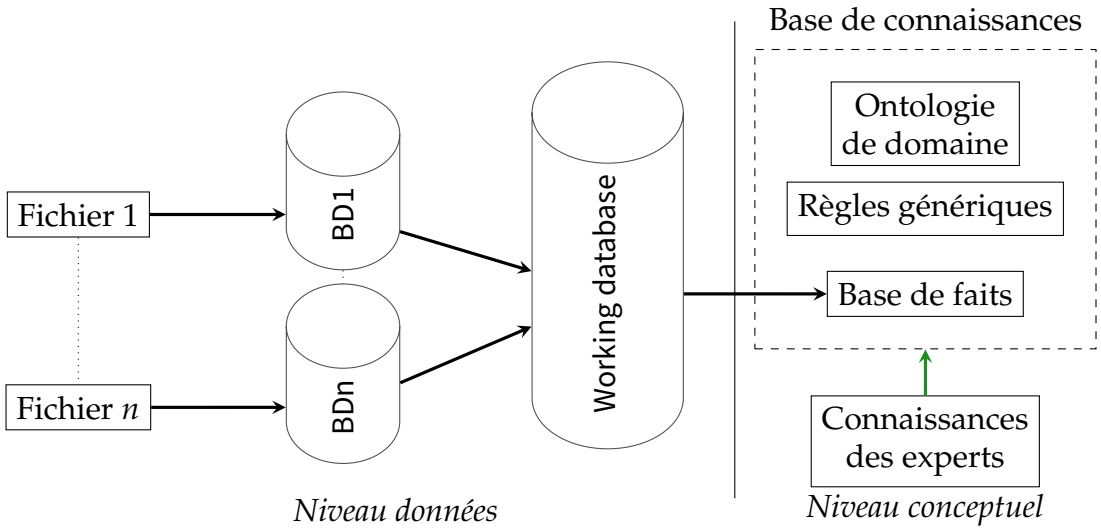


FIGURE 3.1 – Architecture de l'outil

3.1 Architecture générale de l'outil

L'architecture de l'outil (voir Figure 3.1) est fortement inspirée de celle d'OBDA. En effet, comme nous pouvons le voir, nous avons, comme dans le cas d'OBDA, trois niveaux : le niveau des *données* (composé de sources de données), le niveau *conceptuel* (qui contient la base de connaissances) et le niveau des *mappings* (les mappings sont représentés par les flèches noires sur la figure) ; nous verrons les différentes formes de mappings plus en détails plus loin dans ce chapitre (en section 3.3).

Comme le montre la figure, le niveau des données est séparé en deux étapes :

- La première étape permet de passer des sources de données (principalement de fichiers textes, *Fichier 1* à *Fichier n*) à des bases de données relationnelles (*BD1* à *BDn*). Étant donné que les données en agronomie utilisées dans notre étude étaient rendues disponibles en format de fichier texte formaté (de type csv), cette étape était nécessaire afin d'effectuer un nettoyage des données et une standardisation des valeurs permettant la construction d'une base de données (en *MySQL*) contenant des données plus propres pouvant être exploitées automatiquement (cette étape de nettoyage, propre à chaque source de données, est détaillée davantage dans le chapitre 4, section 4.1.5).

- La seconde étape est le passage de plusieurs bases de données à une même base de données de travail (qu'on appelle *working database*). Ce passage permet de sélectionner et d'agréger des valeurs issues de chaque base de

données.

Au niveau conceptuel nous avons une base de connaissances exprimée dans un langage de représentation des connaissances et de raisonnement basé sur des règles. Cette base de connaissances contient deux types de connaissances : (1) des *faits*, qui sont des assertions sur des entités spécifiques (formant ensemble la *base de faits*) ; et (2) des *règles*, qui expriment des connaissances générales pouvant être appliquées aux faits afin d'en créer de nouveaux.

Précisons que la base de faits est alimentée de deux façons : (1) d'une part à partir des données qui sont présentes dans la *working database* (via des mappings) et (2) d'autre part à partir de connaissances qui viennent directement des experts (représentées par une flèche verte sur la figure). Typiquement, les connaissances fournies par les experts sont celles qui décrivent les liens entre des traits, des fonctions et des services (voir chapitre 4, section 4.2 pour plus de détails sur l'acquisition et l'intégration des connaissances expertes).

Par ailleurs, nous distinguons sur la figure deux ensembles de règles. Le premier ensemble est formé de règles qui traduisent l'*ontologie de domaine*. L'ontologie de domaine fournit un vocabulaire lié au domaine d'application en termes de concepts et de relations qui ont un sens pour un utilisateur. Les relations sémantiques entre les éléments du vocabulaire sont spécifiées par des règles simples exprimant principalement les inclusions entre concepts ou entre relations ainsi que les signatures des relations. Le second ensemble est formée de règles, généralement plus complexes, qui effectuent des calculs. Ces règles s'appuient sur les deux types de faits pour estimer la contribution des espèces aux fonctions et services écosystémiques. Elles sont dites *génériques* car elles sont indépendantes de services écosystémiques ou de traits particuliers. Elles peuvent donc être exploitées pour d'autres cas d'étude qui utilisent l'approche traits - fonctions - services.

3.2 Langage de représentation des connaissances

En ce qui concerne le choix d'un formalisme, nous n'avons pas fait de choix *a priori*, mais avons décidé au contraire de partir des connaissances applicatives à représenter afin d'identifier le formalisme le plus adapté, c'est-à-dire permettant d'exprimer ces connaissances d'une façon naturelle, tout en ayant une expressivité "minimale" de façon à limiter la complexité des inférences. L'ontologie de domaine, et les connaissances expertes de façon plus générale, s'expriment naturellement sous la forme de règles logiques. En l'état actuel de notre modélisation, nous considérons des règles Datalog étendues à des fonctions calculées, qui per-

mettent d'intégrer divers calculs (comme par exemple calculer la valeur moyenne d'un ensemble de traits fonctionnels) dans le formalisme logique et un usage restreint de la négation par l'échec (qui permet d'exprimer l'absence de certains faits, nous verrons plus de détails sur la gestion de la négation dans la Section 3.2.3 de ce chapitre).

Pour définir les notions importantes (comme l'application des règles, permettant de construire de nouveaux faits), nous considérons dans un premier temps des règles Datalog classiques (sans les fonctions calculées et sans la négation). Ensuite nous ajouterons les *fonctions calculées* en étendant les notions déjà définies et enfin, ce travail sera étendu à l'intégration de la *négation*.

3.2.1 Datalog

La version classique de Datalog [CGT+89 ; AHV94] est un langage de requêtage pour les bases de données relationnelles. Nous en donnons ici une présentation équivalente, plus adaptée à notre vision "bases de connaissances". Comme dans la Section 2.1.2, nous structurons notre base de connaissances en une base de faits et en une ontologie [1].

La base de faits est un ensemble d'atomes instanciés (comme dans la Section 2.1.2), construite sur un vocabulaire défini de la même façon qu'en Section 2.1.1. L'ontologie est un ensemble de *règles Datalog*, comme nous les avons définies en Section 2.1.5. Le problème fondamental auquel nous nous intéressons est, comme dans la Section 2.1.2 "calculer toutes les réponses à une requête conjonctive dans la base de connaissances".

Cette section est dévolue à la mise en place du cadre formel pour les algorithmes permettant de répondre à cette question. Dans le cas où la base de connaissances est réduite à une base de faits (quand il n'y a pas de règles), nous verrons que ce calcul se réduit à la recherche d'*homomorphismes*. Sinon, nous verrons que ce calcul peut se faire en calculant une *saturation* de la base de faits.

Nous commençons cette section par quelques définitions.

Définition 3.2.1 (Substitution) *Soit X un ensemble de variables et T un ensemble de termes. Une* substitution *σ de X dans T est une application de X dans T.*

Appliquer une substitution σ à une formule F consiste à remplacer dans F chaque occurence d'une variable $X_i \in X$ par son image $\sigma(X_i) \in T$. Le résultat est noté $\sigma(F)$.

Une substitution σ de X dans T sera représentée par un ensemble de couples

1. Nous prenons ontologie à un sens large : il s'agit de l'ontologie de domaine, mais aussi de toutes les connaissances générales qu'on exprime par des règles.

$$\{(X_1, \sigma(X_1)), \ldots, (X_k, \sigma(X_k))\}$$

où les X_i sont les éléments de X et les $\sigma(X_i)$ des termes dans T.

Définition 3.2.2 (Homomorphisme) *Un* homomorphisme *d'un ensemble d'atomes source G vers un ensemble d'atomes cible F est une substitution π de variables(G) (l'ensemble des variables de G) dans termes(F) (l'ensemble des termes de F) telle que $\pi(G) \subseteq F$. On dit que G s'envoie sur F par π.*

L'existence d'un homomomorphisme correspond très exactement à la notion de conséquence logique. En effet, nous pouvons voir un ensemble d'atomes comme une formule logique, qui est la fermeture existentielle de la conjonction des atomes qui le composent. Plus précisément, si F est un ensemble d'atomes, nous notons $\phi(F)$ la conjonction des atomes qui composent F et $\Phi(F)$ la fermeture existentielle de $\phi(F)$. Par exemple, si $F = \{p(a, X), q(X, Y, b)\}$, nous avons $\phi(F) = p(a, X) \wedge q(X, Y, b)$ et $\Phi(F) = \exists X \exists Y p(a, X) \wedge q(X, Y, b)$. Par la suite, et lorsqu'il n'y a pas d'ambiguité, nous confondrons un ensemble d'atomes F avec sa traduction logique $\Phi(F)$ et pourrons écrire, par exemple, $F \models G$ (G est conséquence logique de F), lorsque F et G sont deux ensembles d'atomes. Le résultat suivant est bien connu :

Théorème 3.2.1 *Soit F et G deux ensembles d'atomes. $F \models G$ si et seulement s'il existe un homomorphisme de G dans F.*

Définition 3.2.3 (Corps) *Un corps C est un ensemble d'atomes (pas nécessairement instanciés).*

Définition 3.2.4 (Réponse à un corps) *La* réponse *à un corps C dans un ensemble d'atomes F est un homomorphisme de C dans F (qui envoie donc les variables de C dans les termes de F).*

Définition 3.2.5 (Requête) *Une* requête *est une paire $(\vec{X}, C)$ où C est un corps et $\vec{X}$ un sous-ensemble ordonné des variables de C (appelées variables réponses).*

On notera aussi une requête sous la forme $Q(\vec{X}) = C$. Par exemple, la requête $((X, Z), \{p(X, Y), q(Y, Z)\})$ sera aussi notée $Q(X, Z) = \exists Y\ p(X, Y), q(Y, Z)$.

Définition 3.2.6 (Réponse à une requête) *Soit $\mathcal{F}$ une base de faits et $Q = (\vec{X}, C)$ une requête avec $\vec{X} = (X_1, \ldots, X_k)$. Le tuple de constantes $\vec{a} = (a_1, \ldots, a_k)$ est une réponse à Q dans $\mathcal{F}$ si il existe une réponse π à C dans $\mathcal{F}$ telle que pour tout X_i, $\pi(X_i) = a_i$.*

En d'autres termes, $\vec{a}$ est une réponse à la requête $(\vec{X}, C)$ dans $\mathcal{F}$ si et seulement si, en notant σ la substitution qui envoie chaque X_i sur a_i, on a $\mathcal{F} \models \sigma(C)$.

Nous rappelons ci-dessous la forme générale d'une règle comme un couple *(corps, tête)* (voir aussi la section 2.1.5). Dans les extensions de Datalog, une règle sera toujours vue de cette façon, même si nous définissons le corps et la tête de façon plus complexe.

Définition 3.2.7 (Règle) *Une règle est un couple (C, T) où C est un corps et T un ensemble d'atomes appelé* tête*, tel que les variables de la tête sont un sous-ensemble des variables positives du corps.*

En Datalog, le corps et la tête d'une règle sont des ensembles d'atomes. Nous écrirons par la suite $C \to T$ pour la règle (C, T). A chaque règle $R = C \to T$, nous associons la formule logique $\Phi(R)$ obtenue par la fermeture universelle de la formule $\phi(C) \to \phi(T)$. De la même façon que pour les ensembles d'atomes, nous confondrons, quand il n'y a pas d'ambiguïté, une règle avec sa traduction logique. Par exemple, si $R = \Big(\{p(X,Y), q(Y,Z)\}, \{r(X,Z)\}\Big)$, nous avons $\phi(R) = p(X,Y) \wedge q(Y,Z) \to r(X,Z)$ et $\Phi(R) = \forall X \forall Y \forall Z \ (p(X,Y) \wedge q(Y,Z) \to r(X,Z))$.

De façon intuitive, l'application d'une règle $C \to T$ sur une base de faits $\mathcal{F}$ sera toujours exprimée de la façon suivante : si π est une réponse à C dans $\mathcal{F}$, alors on peut calculer la base de faits T' qui est *l'évaluation de T dans le contexte de* π. Alors le résultat de cette application de règle sera la base de faits $\mathcal{F} \cup T'$. Cette abstraction du mécanisme d'application de règles nous permet d'avoir une définition unique pour tous les langages que nous considérons dans ce chapitre : il nous suffit juste de définir ce qu'est une réponse à un corps (voir définition 3.2.4 pour le cas de Datalog) et le mécanisme d'évaluation de la tête (qui n'est dans le cas Datalog que l'application d'une substitution).

Définition 3.2.8 (Évaluation de la tête d'une règle - cas Datalog) *Soit $C \to T$ une règle Datalog et π une réponse à C dans une base de faits $\mathcal{F}$. Alors* l'évaluation de T dans le contexte de π *est une base de faits* $\mathsf{evaluation}(T, \pi) = \pi(T)$.

Définition 3.2.9 (Application d'une règle) *Si $R : C \to T$ est une règle et π est une réponse à C dans une base de faits $\mathcal{F}$ alors l'application de R sur $\mathcal{F}$ suivant π produit une base de faits $\mathcal{F}' = \mathcal{F} \cup evaluation(T, \pi)$.*

Exemple 3.2.1 *Soit la base de faits $\mathcal{F} = \{A(a), B(b)\}$ et la règle $(R) : A(X) \wedge B(Y) \to C(X,Y)$.*

La règle R s'applique par l'homomorphisme $\pi = \{(X, a), (Y, b)\}$. *L'évaluation de la tête dans le contexte de* π *est :* $evaluation(C(X, Y), \pi) = \{C(a, b)\}$. *On obtient comme nouvelle base de faits* $\mathcal{F}' = \{A(a), B(b), C(a, b)\}$.

Remarquons que, puisque les variables de la tête d'une règle sont des variables (positives) du corps, et que les réponses au corps dans une base de faits (instanciée) ne contiennent que des constantes, l'évaluation de la tête retournera des atomes instanciés et l'application de la règle produira bien une base de faits (instanciée).

On dit qu'une application de règle produisant $\mathcal{F}' = \mathcal{F} \cup evaluation(T, \pi)$ est *utile* lorsque $\mathcal{F} \neq \mathcal{F}'$, c'est à dire lorsqu'il existe un atome dans $evaluation(T, \pi)$ qui n'est pas dans $\mathcal{F}$.

3.2.1.1 Chaînage avant

Le chaînage avant (ou *chase* dans la littérature de bases de données) est une méthode de déduction qui sature une base de faits en y ajoutant tous les atomes que l'on peut obtenir par une séquence d'applications de règles. La suite de base de faits obtenue par une séquence d'applications de règles est appelée une dérivation.

Définition 3.2.10 (Dérivation) *Soit* $\mathcal{F}$ *une base de faits et* $\mathcal{R}$ *un ensemble de règles. Une* dérivation *(à partir de* $\mathcal{F}$*) est une suite* $(\mathcal{F}_0 = \mathcal{F}), \mathcal{F}_1, \ldots, \mathcal{F}_n$ *telle que pour tout* $0 < i \leq n$, *il existe une règle* $(R) : C \rightarrow T$ *de* $\mathcal{R}$ *et une réponse* π *à* C *dans* $\mathcal{F}_{i-1}$ *avec* $\mathcal{F}_i = \mathcal{F}_{i-1} \cup evaluation(T, \pi)$.

Une dérivation $(\mathcal{F}_0, \ldots, \mathcal{F}_n)$ est *complète* si aucune application de règle sur $\mathcal{F}_n$ ne produit de nouvel atome, c'est-à-dire aucune application sur $\mathcal{F}_n$ n'est utile ; puisque en Datalog il existe toujours une dérivation complète finie. Nous appelons *saturation* de $\mathcal{F}$ (notée $\mathcal{F}^*$) par $\mathcal{R}$ le dernier élément de la dérivation $\mathcal{F}_n$ ($\mathcal{F}_n = \mathcal{F}^*$).

La saturation d'une base de faits nous permet de capturer la notion de conséquence sémantique.

Théorème 3.2.2 *Soit* $\mathcal{F}$ *une base de faits,* $\mathcal{R}$ *un ensemble de règles Datalog et* Q *un ensemble d'atomes. Alors* $\mathcal{F}, \mathcal{R} \models Q$ *ssi il existe un homomorphisme de* Q *dans* $\mathcal{F}^*$, *où* $\mathcal{F}^*$ *est la saturation de* $\mathcal{F}$ *par* $\mathcal{R}$.

En effet, $\mathcal{F}^*$ correspond à un modèle de la base de connaissances et c'est même l'intersection de tous les modèles de la base de connaissances. C'est pourquoi

pour tester si un ensemble d'atomes (ou une requête conjonctive booléenne) est vrai dans tous les modèles de la base de connaissances, il suffit de tester s'il est vrai dans le modèle associé à $\mathcal{F}^*$.

Nous pouvons donc maintenant définir la réponse à une requête dans une base de connaissances composée dans une base de faits et d'un ensemble de règles.

Définition 3.2.11 (Réponse à une requête dans une KB) *Soit $\mathcal{K} = (\mathcal{F}, \mathcal{R})$ une base de connaissances composée d'une base de faits $\mathcal{F}$ et d'un ensemble de règles $\mathcal{R}$, et Q une requête. Une réponse à Q dans $\mathcal{K}$ est une réponse à Q dans $\mathcal{F}^*$, la saturation de $\mathcal{F}$ par $\mathcal{R}$.*

3.2.2 Datalog avec des fonctions calculées

Nous souhaitons maintenant ajouter à notre langage des fonctions calculées et des prédicats calculés dont les arguments sont des littéraux. Considérons par exemple la règle :

$$\text{valeur}(X, \text{base1}, V1), \text{valeur}(X, \text{base2}, V2) \rightarrow \text{valeurFinale}(X, \text{moyenne}(V1, V2))$$

Cette règle veut dire que si on a lu dans base1 la valeur $V1$ pour le critère X et la valeur $V2$ dans la base2 pour le même critère, on considèrera comme valeur finale pour ce critère la moyenne de $V1$ et $V2$.

Le corps de cette règle récupère des substitutions de $\{X, V1, V2\}$ ($V1$ et $V2$ sont les valeurs de l'entité X, respectivement dans les bases base1 et base2), et son application suivant une de ces substitutions est censée créer un atome représentant la valeur finale de cette entité, calculée à partir de la moyenne de $V1$ et de $V2$.

Afin d'obtenir ce comportement, nous ne pourrons pas considérer la fonction moyenne comme une fonction logique quelconque. Il nous faudra fixer son interprétation, par exemple, en indiquant que cette fonction moyenne doit s'interpréter par la méthode `average` dans le langage de programmation utilisé (et nous dirons que *l'interprétation* de moyenne est `average`).

Or cette méthode `average` ne pourra être appelée que sur des objets de notre langage de programmation, et ces objets devront avoir un certain type (par exemple `Float`). Et l'application de cette méthode sur ces objets produira également un objet de type `Float`. Cependant, notre langage logique ne manipule pas des objets d'un langage de programmation, mais des termes. Ces termes devront pouvoir

s'interpréter par des objets du type adéquat afin de pouvoir appliquer la méthode `average`, et l'objet produit par l'application de cette méthode devra pouvoir être *représenté* par un terme.

Nous présentons ci-dessous une façon de mettre en correspondance ces termes interprétables avec les objets d'un langage de programmation par l'intermédiaire de *littéraux* typés par des *datatypes*. Cette façon de faire a été conçue dans le cadre du développement du logiciel *Graal* (sur lequel est basée notre implémentation) mais n'est que partiellement implémentée dans Graal pour l'instant : Graal dispose de fonctions et prédicats calculés mais la notion de datatype n'est pas encore implémentée. La modélisation du cas d'étude que nous présentons dans les chapitres suivants n'utilise donc pas explicitement les datatypes.

Il a été décidé d'utiliser, pour les constantes et fonctions interprétables dans un langage de programmation, la notion de *littéral* et de *datatype* telle qu'elle a été définie dans le langage RDF. Pour les fonctions interprétables par une méthode dans un langage de programmation, nous utiliserons les *fonctions calculées* (qui ne sont pas présentes en RDF). De la même façon, nous utiliserons la notion de *prédicat calculé*. Ces nouveaux objets devront être définis dans le vocabulaire, aussi notre premier travail devra être d'étendre sa définition. Par la suite, nous étendrons le langage Datalog pour prendre en compte ces nouveaux objets, en redéfinissant les atomes, les corps et leur réponses, ainsi que les têtes de règles et leur évaluation.

Nous introduirons également des contraintes syntaxiques sur le typage de nos objets afin d'assurer que le mécanisme de raisonnement par saturation ne produise aucune erreur de typage dans le langage de programmation.

3.2.2.1 Vocabulaire

Outre les constantes (dont l'ensemble est noté $\mathcal{C}$), variables (dont l'ensemble est noté $\mathcal{V}$) et prédicats, maintenant appelés *prédicats standards*, (dont l'ensemble est noté $\mathcal{P}_s$) considérés jusqu'à présent, notre vocabulaire contient maintenant de nouveaux objets :

- un ensemble $\mathcal{T}$ de *datatypes*, chaque datatype étant *interprété* par un type de notre langage de programmation ;
- un ensemble $\mathcal{S}$ de *représentations* (qui seront utilisées pour représenter les objets du langage de programmation) ;
- un ensemble $\mathcal{F}$ de fonctions calculées, chacune étant interprétée par une méthode de notre langage de programmation ;
- et un ensemble $\mathcal{P}_c$ de *prédicats calculés*, chacun étant interprété par une

méthode dont le type de retour est booléen (i.e. la méthode retourne soit vrai, soit faux).

Comme les prédicats standards, les prédicats calculés et les fonctions calculées ont une *arité*. Mais dans ce cas, l'arité sera déterminée par le nombre de paramètres de la méthode qui interprète le prédicat ou la fonction. Un prédicat ou une fonction calculée f d'arité k aura pour *signature* un tuple de datatypes $(d_1, \ldots, d_k)$ de taille k, tel que, pour chaque d_i, l'interprétation de d_i est le type du i-ème argument de la méthode qui interprète f. Nous munissons également chaque prédicat standard d'arité k d'une signature : celle-ci sera un tuple de taille k dont chaque élément est soit un datatype, soit le type particulier _ dont les instances sont les constantes logiques. Enfin, une fonction calculée f aura un *type de retour* : un datatype dont l'interprétation est le type de retour de la méthode qui interprète f.

Exemple 3.2.2 *Les datatypes Float1 et Float2 sont tous deux interprétés par le type* `float`. *La fonction moyenne est interprétée par la méthode* `average`, *dont les paramètres sont* `(float, float)` *et la valeur de retour est* `float`. *Comme cette méthode est d'arité 2, la fonction calculée moyenne est également d'arité 2. Enfin, la signature de moyenne pourra être (Float1, Float2) – ou (Float1, Float1) – et sa valeur de retour pourra être Float2.*

En Datalog, les termes étaient soit des variables, soit des constantes. Les termes sont maintenant soit des termes atomiques, soit des termes fonctionnels que nous définirons un peu plus tard. Les termes atomiques sont soit des variables, soit des constantes, (variables et constantes forment les termes logiques) soit des littéraux.

Définition 3.2.12 (Littéral) *Un littéral est une paire $l = (d, r)$ où d est un datatype de $\mathcal{T}$ et r est une représentation de $\mathcal{S}$, tel que* l'interprétation *de l est un objet o dont le type est l'interprétation de d. Dans ce cas, la* représentation *de o est un littéral $l' = (d, r')$ dont l'interprétation est o.*

Par la suite nous noterons, en adoptant les conventions RDF, $''r''\hat{}\hat{}d$ le littéral (d, r). Ainsi "26/07/2022"^^date, où le datatype date est lié au type `DateTime`, sera interprété comme une instance de la classe `DateTime` correspondant à la date représentée.

Exemple 3.2.3 *$''7''\hat{}\hat{}$Float est un littéral dont l'interprétation est le* `float 7.0`. *La représentation de cet objet pourrait être $''7.0''\hat{}\hat{}$Float.*
Notons que dans notre langage de programmation, l'interprétation de $''7.0''\hat{}\hat{}$Float et celle de $''7''\hat{}\hat{}$Float sont identiques.

Un terme fonctionnel se définit naturellement de façon inductive. Cependant, nous devons nous assurer que le typage soit correct dans la définition d'un tel terme, qui peut contenir des variables (non typées).

Exemple 3.2.4 *Prenons par exemple la fonction calculée repeat dont la signature est (Int, String) et la valeur de retour String, interprétée par une méthode qui concatène N fois une chaîne. Ainsi le terme fonctionnel repeat(N, S) sera correct mais le terme fonctionnel repeat(S, S) ne le sera pas, car aucun littéral substituant S ne pourra avoir à la fois le datatype String et le datatype Int.*

Afin de résoudre ce problème, toutes nos définitions utiliserons inductivement les notions de *type* et de *typevars*, qui encode le type de toutes les variables apparaissant dans la construction de l'objet. Ainsi, le typevars d'un objet sera une application de l'ensemble des variables apparaissant dans l'objet dans l'ensemble des datatypes union $\{\_\}$. Deux typevars seront compatibles s'ils font correspondre le même type aux mêmes variables. On note $typevars1 \subseteq typevars2$ lorsque toute variable de $typevars1$ apparait avec le même type dans $typevars2$.

Définition 3.2.13 (Terme fonctionnel) *Un terme fonctionnel est de la forme*

$$t = f(t_1, \ldots, t_k)$$

où f est une fonction calculée d'arité k et de signature $(d_1, \ldots, d_k)$. Le type de t est le type de retour de f. Chaque t_i est un terme tel que $type(t_i)$ doit être égal à d_i. Les $typevars(t_i)$ doivent être 2 à 2 compatibles, et on définit $typevars(t) = \bigcup_{1 \leq i \leq k} typevars(t_i)$.

Chaque terme t_i peut être :

- *un littéral de type d, auquel cas $type(t_i) = d$ et $typevars(t_i) = \{\}$;*
- *une variable, auquel cas, $type(t_i) = d_i$ et $typevars(t_i) = \{t_i : d_i\}$;*
- *un terme fonctionnel, auquel cas type et typevars sont définis inductivement.*

Notons qu'un terme fonctionnel peut contenir des variables et des littéraux, mais jamais de constantes logiques.

Exemple 3.2.5 *Soit le terme fonctionnel suivant : $w = f(u, v)$ avec u et v des termes fonctionnels tels que $u = g(X, Y)$ avec pour signature $(Bool, Int)$ et $v = h(''7''\hat{}\hat{}Int, Y)$ avec pour signature (Int, Int). f, g et h définissent des fonctions calculées d'arité 2 ayant pour type de retour Int, Int et $Bool$ respectivement.*
Notre objectif est de calculer $typevars(w)$. Pour cela commençons par calculer $typevars(u)$ et $typevars(v)$.

On commence par $u = g(X, Y)$:

- *X étant une variable,* $typevars(X) = \{X : signature(g)[1] = Bool\}$;
- *Y étant également une variable,* $typevars(Y) = \{Y : signature(g)[2] = Int\}$.

$typevars(X) \cap typevars(Y) =$ *donc pas de problèmes de compatibilité.*

Donc $typevars(u) = \{X : Bool, Y : Int\}$.

Passons à $v = h(''7''\hat{}\hat{}Int, Y)$:

- $''7''\hat{}\hat{}Int$ *est un littéral, donc,* $typevars(''7''\hat{}\hat{}Int) = \{\}$;
- *Y est une variable donc* $typevars(Y) = \{Y : Int\}$.

$typevars(7) \cap typevars(Y) =$ *donc pas de problèmes de compatibilité.*

Donc $typevars(v) = \{Y : Int\}$.

Pour finir, on passe à $w = f(u, v)$:

- *u est un terme fonctionnel,* $typevars(u) = \{X : Bool, Y : Int\}$;
- *v est un terme fonctionnel,* $typevars(v) = \{Y : Int\}$;

$typevars(u) \cap typevars(v) = \{Y : Int\}$ *(Y apparait dans les* 2 *cas comme étant de même type,* Int*).*

Donc $typevars(w) = \{X : Bool, Y : Int\}$

Exemple 3.2.6 *Reprenons l'exemple précédent en modifiant maintenant la signature de* $u = g(X, Y)$ *qui est désormais* $(Bool, Bool)$*.*

Nous gardons $w = f(u, v)$ *avec u et v des termes fonctionnels et* $v = h(''7''\hat{}\hat{}Int, Y)$ *avec pour signature* (Int, Int)*.*

Dans ce cas $typevars(u)$ *devient* $typevars(u) = \{X : Bool, Y : Bool\}$*. Et* $typevars(v)$ *ne change pas :* $typevars(v) = \{Y : Int\}$.

Passons à $w = f(u, v)$:

- *u est un terme fonctionnel,* $typevars(u) = \{X : Bool, Y : Bool\}$;
- *v est un terme fonctionnel,* $typevars(v) = \{Y : Int\}$;

Dans ce cas, nous voyons que les types ne sont pas compatibles, en effet, Y est de type Bool dans un cas et Int dans l'autre.

Un terme fonctionnel t est *instancié* lorsque $typevars(t) = \{\}$ (c'est-à-dire que t ne contient aucune variable, ou que t ne contient que des littéraux). Dans ce cas, l'interprétation de t peut se définir de la façon suivante.

Définition 3.2.14 (Interprétation d'un terme fonctionnel) *L'interprétation d'un terme fonctionnel instancié* $t = f(t_1, \ldots, t_k)$ *est un objet (de notre langage de programmation) calculé inductivement de la façon suivante : soit F la méthode (dont le type de retour est D) qui interprète* f *et* $(T_1, \ldots, T_k)$ *les objets qui interprètent* $(t_1, \ldots, t_k)$*. Alors l'interprétation de t est l'objet de type D obtenu en appliquant F sur les objets* $(T_1, \ldots, T_k)$.

Exemple 3.2.7 *Supposons un terme fonctionnel* $t = f(g(''4''\hat{}\hat{}Int,'' 3''\hat{}\hat{}Int),'' 7''\hat{}\hat{}Int)$ *où* f *et* g *sont deux fonctions calculées qui sont liées aux méthodes de soustraction et d'addition (on suppose que ces deux méthodes retournent un objet de type* Int*).*

Pour interpréter t*, nous interprétons d'abord* $g(''4''\hat{}\hat{}Int,'' 3''\hat{}\hat{}Int)$ *:*

$''4''\hat{}\hat{}Int$ *et* $''3''\hat{}\hat{}Int$ *étant des littéraux, ils sont remplacés par les valeurs* 4 *et* 3 *respectivement, nous obtenons donc* $g(4,3)$*.*

g *étant une fonction, liée à la méthode d'addition, donc* $g(4,3)$ *retourne la valeur* 7*.*

Les mêmes étapes sont faites pour $f(7,'' 7''\hat{}\hat{}Int)$ *(avec* f *liée à la soustraction). On obtient donc que l'interprétation de* f *est* 0*.*

Notons que nous ne pouvons interpréter que les termes fonctionnels instanciés, et que notre définition (utilisant les typevars) assure que l'interprétation d'un terme fonctionnel instancié pourra être calculé sans que notre langage de programmation ne retourne d'erreur de typage.

Dans la suite, pour simplifier et lorsqu'il n'y a pas d'ambiguïté nous notons les littéraux sans préciser leur type. Par exemple, $''4''\hat{}\hat{}Int$ sera directement noté 4, $''4.0''\hat{}\hat{}Float$ sera noté 4.0, $''Alice''\hat{}\hat{}String$ sera noté $''Alice''$, etc

3.2.2.2 $\mathcal{L}$-atomes et $\mathcal{L}$-corps

En Datalog, nous nous sommes intéressés à des ensembles d'atomes dont les termes étaient toujours des termes logiques. Nous souhaitons maintenant y intégrer notre nouveau vocabulaire : littéraux, fonctions et prédicats calculés. La notion de *typevars* est très importante pour assurer le bon typage des formules que nous allons utiliser. Nous l'étendons donc aux atomes et aux conjonctions d'atomes de la façon suivante :

Définition 3.2.15 ($\mathcal{L}$-atome) *Un* $\mathcal{L}$*-atome est de la forme* $a = p(t_1,\ldots,t_k)$ *où* p *est un prédicat standard (dans ce cas* a *est dit standard) ou calculé (dans ce cas* a *est dit calculé) d'arité* k *et de signature* $(d_1,\ldots,d_k)$ *et les* t_i *sont des termes tels que* $type(t_i)$ *doit être égal à* d_i *et dont les typevars sont 2 à 2 compatibles. On définit* $typevars(a) = \bigcup_{1\leq i\leq k} typevars(t_i)$*.*

Chaque t_i *peut être :*

- *une constante, auquel cas* $type(t_i) =$ "_" *et* $typevars(t_i) = \{\}$ *;*
- *une variable, auquel cas* $type(t_i) = d_i$ *et* $typevars(t_i) = \{t_i : d_i\}$ *;*
- *un littéral de type* d*, auquel cas* $type(t_i) = d$ *et* $typevars(t_i) = \{\}$ *;*
- *un terme fonctionnel, auquel cas nous utilisons les définitions de type et typevars de la Définition 3.2.13.*

Si $a = p(t_1, \ldots, t_k)$ est un atome calculé et instancié, tel que P est l'interprétation de p et $(T_1, \ldots, T_k)$ sont les interprétations de $(t_1, \ldots, t_k)$, alors l'interprétation de a est le booléen obtenu par application de P sur les paramètres $(T_1, \ldots, T_k)$.

Si $A = \{a_1, \ldots, a_p\}$ est un ensemble d'atomes, alors les $typevars(a_i)$ doivent être 2 à 2 compatibles et on définit $typevars(A) = \bigcup_{1 \leq i \leq p} typevars(a_i)$.

L'interprétation d'un ensemble d'atomes calculés et instanciés est vraie si l'interprétation de chacun de ses atomes est vraie, et faux sinon.

Notons ici aussi que l'on ne peut interpréter que les ensembles d'atomes calculés *et instanciés*. Ici aussi, les contraintes syntaxiques imposées par les typevars font que le calcul de cette interprétation se fera toujours sans erreur de typage dans le langage de programmation.

Définition 3.2.16 ($\mathcal{L}$-fait) *Un $\mathcal{L}$-fait est défini comme étant une conjonction de $\mathcal{L}$-atomes standards sans variables ni termes fonctionnels.*

En d'autres termes, un $\mathcal{L}$-fait est, comme en Datalog, un ensemble d'atomes (standards) instanciés mais qui peuvent maintenant contenir des littéraux, à condition de respecter la signature des prédicats.

Nous définissons ici un $\mathcal{L}$-corps qui servira de base à la définition des $\mathcal{L}$-requêtes et à celle des $\mathcal{L}$-règles.

Définition 3.2.17 ($\mathcal{L}$-corps) *Un $\mathcal{L}$-corps est l'union de deux ensembles d'atomes :*

- *la* partie standard *PS est un ensemble de $\mathcal{L}$-atomes standard sans termes fonctionnels ;*
- *la* partie calculée *PC est un ensemble de $\mathcal{L}$-atomes calculés tel $typevars(PS) \subseteq typevars(PC)$.*

Nous devons maintenant définir ce qu'est la réponse à un $\mathcal{L}$-corps dans un $\mathcal{L}$-fait. Intuitivement, nous voudrions que ce soit un homomorphisme de la partie standard qui satisfait la partie calculée. Cependant, comme le montre l'exemple suivant, il nous faut tout d'abord légèrement modifier notre notion d'homomorphisme.

Exemple 3.2.8 *Soit $F = \{p(a,''2.0''\hat{}\hat{}Float)\}$ un $\mathcal{L}$-fait et $C = \{p(X,''2''\hat{}\hat{}Float)\}$ un $\mathcal{L}$-corps (ne contenant qu'une partie standard). La substitution $\sigma = \{X : a\}$ n'est pas un homomorphisme de C dans F car $\sigma(C) \not\subseteq F$.*

Le problème avec la notion actuelle d'homomorphisme est que les littéraux $''2.0''\hat{}\hat{}Float$ et $''2''\hat{}\hat{}Float$ sont considérés comme différents alors qu'ils devraient

avoir la même interprétation. Nous devons donc remplacer la notion d'homomorphisme par celle de $\mathcal{L}$-homomorphisme, qui présuppose une normalisation des littéraux.

Définition 3.2.18 ($\mathcal{L}$-homomorphisme) *Soit F un $\mathcal{L}$-fait et CS la partie standard d'un $\mathcal{L}$-corps. Un $\mathcal{L}$-homomorphisme de CS dans F est un homomorphisme de $norm(CS)$ dans $norm(F)$, où $norm(A)$ remplace dans un ensemble d'atomes A chaque occurence d'un littéral l par sa normalisation, qui est la représentation de son interprétation.*

Remarquons que si π est un $\mathcal{L}$-homomorphisme de la partie standard d'un $\mathcal{L}$-corps $PS \cup PC$ dans un $\mathcal{L}$-fait, alors $\pi(PC)$ est une partie calculée instanciée : nous pouvons donc l'interpréter.

Définition 3.2.19 (Réponse à un $\mathcal{L}$-corps) *Soit F un $\mathcal{L}$-fait et $C = PS \cup PC$ un $\mathcal{L}$-corps. Une réponse à C dans F est un $\mathcal{L}$-homomorphisme π de PS dans F tel que π* satisfait *PC, c'est à dire que l'interprétation de $\pi(PC)$ retourne vrai.*

Exemple 3.2.9 *$F = \{valeurBase(trait, base1, 0.3), valeurBase(trait, base2, 0.8)\}$ et $C = \{valeurBase(T, B1, V1), valeurBase(T, B2, V2), moy(V1, V2) > 0.5\}$ un $\mathcal{L}$-corps ayant pour partie calculée $moy(V1, V2) > 0.5$.*
On considère le $\mathcal{L}$-homomorphisme π allant de la partie standard de C dans F, tel que $\pi = \{(T, trait), (B1, base1), (B2, base2), (V1, 0.3), (V2, 0.8)\}$. π satisfait sa partie calculée, en effet $\pi(moy(V1, V2)) = moy(0.3, 0.8) = 0.55 > 0.5$.

Ayant ainsi défini un $\mathcal{L}$-corps et ses réponses dans un $\mathcal{L}$-fait, nous définissons une $\mathcal{L}$-requête à partir d'un $\mathcal{L}$-corps de la même manière que la requête était définie à partir d'un corps dans la définition 3.2.5. Les réponses à cette $\mathcal{L}$-requête sont définies à partir des réponses au $\mathcal{L}$-corps, en suivant la définition 3.2.6.

Enfin, notons que nous ne donnons ici aucun résultat de correspondence entre l'existence d'une réponse et la conséquence sémantique, comme nous avions fait dans le théorème 3.2.1. En effet, il faudrait pour celà modifier les interprétations et la définition de modèle pour prendre en compte nos nouveaux objets (littéraux, fonctions et prédicats calculés), ce qui sortirait du cadre de cette présentation.

3.2.2.3 $\mathcal{L}$-règles

Définition 3.2.20 ($\mathcal{L}$-règle) *Une $\mathcal{L}$-règle est de la forme $C \rightarrow T$ où C est un $\mathcal{L}$-corps et T une conjonction d'atomes sans prédicats calculés (mais pouvant contenir des fonctions calculées) telle que $typevars(T) \subseteq typevars(C)$.*

Exemple 3.2.10 *Soit la règle* (R) : $aPourAge(X, A_1), aPourAge(Y, A_2), A_1 > 0, A_2 > 0 \rightarrow aPourMoyenneDage(X, Y, moy(A_1, A_2))$.

Cette règle calcule l'âge moyen (en utilisant moy qui est liée à une fonction calculant la moyenne) de 2 entités (X et Y) qui ont pour âges respectifs (A_1 et A_2) tels que A_1 et A_2 sont obligatoirement positifs.

Si t est un terme sans variable, alors nous définissons l'évaluation de t de la façon suivante :

- si t est un terme fonctionnel, alors l'evaluation de t est la représentation de l'interprétation de t ;
- sinon, l'évaluation de t sera le terme t lui même.

Si $a = p(t_1, \ldots, t_k)$ un atome standard instancié (c'est-à-dire qui ne contient que des littéraux, des constantes et des termes fonctionnels sans variables), alors son évaluation, $evaluation(a) = p(evaluation(t_1), \ldots, evaluation(t_k))$, est un atome standard instancié sans symbole fonctionnel (qui peut donc être rajouté dans une base de faits). Si A est un ensemble d'atomes standards instanciés, alors $evaluation(A)$ est l'ensemble des atomes $evaluation(a)$ où a est un atome de A.

Remarquons maintenant que si $C \rightarrow T$ est une $\mathcal{L}$-règle, et π est une réponse à C dans un $\mathcal{L}$-fait F, alors $\pi(T)$ est un ensemble d'atomes standards instanciés, et est donc évaluable comme défini au paragraphe précédent. Cette évaluation produit un $\mathcal{L}$-fait. Cette remarque nous permet donc de définir la notion d'évaluation de la tête d'une $\mathcal{L}$-règle :

Définition 3.2.21 (Évaluation de la tête d'une $\mathcal{L}$-règle) *Soit F un $\mathcal{L}$-fait, $C \rightarrow T$ une $\mathcal{L}$-règle et π une réponse à C dans F. Alors l'évaluation de T dans le contexte de π est un $\mathcal{L}$-fait obtenu par l'évaluation de $\pi(T)$.*

Ayant ainsi défini le mécanisme d'évaluation d'une tête de règle dans le contexte d'une réponse, nous pouvons définir l'application d'une $\mathcal{L}$-règle comme en section précédente, définition 3.2.9.

Par la suite, les notions de dérivation, de saturation et de réponse à une requête dans une base de connaissances définies à la section précédente peuvent s'étendre naturellement en considérant les fonctions calculées.

3.2.3 Datalog avec négation

Pour terminer sur le formalisme, nous ajoutons à notre langage la négation (notée pour l'instant $\neg$). L'intégration de la négation nous permettra d'exprimer des règles de la forme :

$$estParentDe(X, Y) \wedge \neg estPèreDe(X, Y) \rightarrow estMèreDe(X, Y))$$

signifiant que "si X est un parent de Y mais n'est pas son père, alors X est la mère de Y".

Dans le domaine de la représentation des connaissances, il existe deux hypothèses sur la façon dont les connaissances sont décrites, avec une différence dans la manière de gérer la négation, on parle de *l'hypothèse du monde ouvert* et *l'hypothèse du monde clos*.

3.2.3.1 Hypothèses du monde ouvert et du monde clos

Dans l'hypothèse du monde ouvert, la négation est celle de la logique classique. En effet, elle nous dit que "ce n'est pas parce qu'on ne connait pas une information que cette information est fausse". On suppose qu'on a une connaissance *incomplète* du monde. Considérons un atome instancié $P(a)$:

- "$P(a)$ est vrai" si $P(a)$ est déductible de la base.
- "$P(a)$ est faux" si $\neg P(a)$ est déductible de la base.
- "La valeur de vérité de $P(a)$ est inconnue" si ni $P(a)$ ni $\neg P(a)$ ne sont déductibles de la base.

A contrario, l'hypothèse du monde clos suppose qu'on a une connaissance complète du monde. Un fait est considéré comme faux si on échoue à montrer qu'il est vrai. Autrement dit, tout ce qui est vrai est soit inclus dans la base de faits soit déductible de celle-ci. En d'autres termes, c'est la négation par l'absence qui est considérée, et on la note par *not* pour la différencier de la négation en monde ouvert. Considérons un atome instancié $P(a)$:

- "$P(a)$ est vrai" si $P(a)$ est présent dans la base de faits ou déductible de la base.
- "$P(a)$ est faux" si $P(a)$ n'est pas présent dans la base de faits et n'est pas déductible de la base.

Prenons un exemple pour illustrer les différences entre ces deux hypothèses. On considère la base de faits $\mathcal{F} = \{A(a), B(a), C(a)\}$. Que nous dit cette base sur l'atome instancié $D(a)$? En monde *ouvert*, rien : ni $D(a)$ ni $\neg D(a)$ n'étant présent dans la base, on considère que $D(a)$ peut être vrai ou faux. Par contre, en monde *fermé*, $D(a)$ est faux puisqu'il est absent de $\mathcal{F}$.

La négation du monde ouvert est complexe à traiter et s'avère souvent moins utile que la négation du monde clos. Comme de nombreux langages à base de règles, Datalog considère la négation du monde clos, aussi appelée dans d'autres

langages négation par l'échec ou négation par défaut.

3.2.3.2 $\mathcal{L}$-corps et $\mathcal{L}$-règles avec négation

Revenons à notre formalisme pour identifier les notions qui changent et les adapter en prenant en compte la négation. Nous remarquons que notre vocabulaire ainsi que nos faits ne changent pas (on considère toujours les $\mathcal{L}$-faits). Par contre, la définition d'un $\mathcal{L}$-corps est modifiée comme suit :

Définition 3.2.22 ($\mathcal{L}$-corps avec négation) *Un $\mathcal{L}$-corps avec négation est de la forme $C = C^+, not\, C_1^-, \ldots, not\, C_k^-$ où $C^+ = (PS^+, PC^+)$ est un $\mathcal{L}$-corps appelé* partie positive *et chaque C_i^- est un ensemble de $\mathcal{L}$-atomes appelé une* partie négative *tel que $PS^+ \cup C_i^-$ est un $\mathcal{L} - corps$. Les variables de PS^+ sont appelées* variables positives *de C.*

Remarquons que si $C = C^+, not\, C_1^-, \ldots, not\, C_k^-$ est un $\mathcal{L}$-corps avec négation et π est une réponse à C^+ dans un $\mathcal{L}$-fait F, alors pour chaque partie négative C_i^-, $\pi(C_i^-)$ est bien un $\mathcal{L}$-corps. En effet, toutes les variables de la partie calculée de C_i^- apparaissaient soit dans sa partie standard, soit dans la partie standard PS^+ de C^+. La réponse π ayant substitué toutes les variables de PS^+ par des constantes ou des littéraux, les variables de la partie calculée de $\pi(C_i^-)$ sont maintenant nécessairement dans sa partie positive, et $\pi(C_i^-)$ est bien un $\mathcal{L}$-corps. Nous pouvons donc calculer les réponses à $\pi(C_i^-)$, ce qui justifie la définition suivante :

Définition 3.2.23 (Réponse à un $\mathcal{L}$-corps avec négation) *Une réponse à un $\mathcal{L}$-corps avec négation $C = C^+, not\, C_1^-, \ldots, not\, C_k^-$ dans un $\mathcal{L}$-fait F est une réponse π à C^+ dans F telle que, pour chaque C_i^-, $\pi(C_i^-)$ n'a pas de réponse dans F.*

Dans les définitions impliquant la notion de corps, nous remplaçons celui-ci par un $\mathcal{L}$-corps avec négation. Une $\mathcal{L}$-requête devient un $\mathcal{L}$-corps avec négation ayant des variables réponses qui doivent être des variables *positives*. En effet, nous ne souhaitons pas pouvoir répondre à une requête de la forme : $Q(X) = not\ \text{adulte}(X)$ qui énumérerait toutes les entités qui ne sont pas des adultes, l'ensemble dans lequel prendre ces entités n'étant pas bien défini. Par contre, une requête de la forme $Q(X) = \text{personne}(X) \wedge not\ \text{adulte}(X)$, qui demande les personnes qui ne sont pas des adultes, est admissible puisque X est ici une variable positive. La réponse à une $\mathcal{L}$-requête avec négation est définie comme dans définition 3.2.6.

Définition 3.2.24 ($\mathcal{L}$-règle avec négation) *Une $\mathcal{L}$-règle avec négation est de la forme $C \rightarrow T$ où C est un $\mathcal{L}$-corps avec négation de partie positive C^+ et T est la tête d'une $\mathcal{L}$-règle (définition 3.2.20) tels que $typevars(T) \subseteq typevars(C^+)$.*

Comme la tête d'une $\mathcal{L}$-règle avec négation est identique à la tête d'une $\mathcal{L}$-règle, la définition de son évaluation dans le contexte d'une réponse reste identique (définition 3.2.21). Nous conservons ainsi les définitions d'application de règle, de dérivation, de saturation et de réponse à une requête dans une base de connaissances. Cependant, comme nous allons le voir dans la section suivante, ces définitions ne suffisent pas à définir la sémantique d'un ensemble de règles avec négation.

3.2.3.3 Problèmes posés par la négation par défaut

Le problème fondamental qui se pose est celui de la *persistance* d'une dérivation. Intuitivement, une dérivation est persistante si toute application de règle effectuée dans la dérivation est encore "valide" sur le résultat final. Formellement, $D = \mathcal{F}_0, \ldots, \mathcal{F}_n$ est *persistante* si pour toute application d'une règle $C \rightarrow T$ dans D où π est la réponse à C, π reste une réponse à C dans $\mathcal{F}_n$.

Autrement dit, si une application de règle a nécessité l'absence d'un fait à une certaine étape i, ce fait n'est pas présent non plus dans la base de faits saturée. Ce qui n'est pas toujours le cas. En effet, si on considère la base de faits $\mathcal{F} = \{A(a)\}$ et les règles suivantes :

$$(R_1) : \forall X \; (A(X) \wedge not \; B(X) \rightarrow C(X))$$

$$(R_2) : \forall X \; (C(X) \rightarrow B(X))$$

nous voyons que d'abord la règle R_1 s'applique et produit le fait $C(a)$ qui est ajouté à la base de faits. R_2 s'applique à son tour (pour assurer une dérivation complète) et produit le fait $B(a)$, qu'on ajoute à la base de faits. Le problème qui se pose est que l'application de R_1 part de l'hypothèse que $B(a)$ est "faux" alors qu'il a été déduit par la suite (en appliquant R_2). C'est pour cette raison, que nous nous intéressons par la suite à des dérivations non seulement complètes mais aussi *persistantes*. Notons que le résultat de telles dérivations est un modèle de la base de connaissances. Pour assurer cette propriété, deux possibilités s'offrent à nous :

(i) soit comme certains langages de programmation logique tel que Answer Set Programming (ASP), on autorise plusieurs mondes possibles, l'idée étant que

chaque dérivation persistante et complète définit un monde possible (appelé *modèle stable* de la base de connaissances). Dans notre exemple précédent, nous n'avons aucune dérivation complète et persistante, donc aucune solution possible. Prenons un autre exemple. On considère maintenant la base de faits $\mathcal{F} = \{A(a)\}$ et les règles suivantes :

$$(R_1) : \forall X\ (A(X) \wedge not\ B(X) \rightarrow C(X))$$

$$(R_2) : \forall X\ (A(X) \wedge not\ C(X) \rightarrow B(X))$$

Si on applique R_1 avant R_2, puisque $A(a)$ est dans la base de faits et $B(a)$ ne l'est pas (donc on considère que $notB(a)$ est vrai) alors on produit $C(a)$. R_2 ne pouvant plus être appliquée, nous avons bien une dérivation persistante et complète dont le résultat est $\{A(a), B(a)\}$.

Dans le même principe si on applique R_2 avant R_1 on produit uniquement le fait $B(a)$. Cette dérivation persistante et complète a pour résultat $\{A(a), B(a)\}$. Nous avons donc 2 solutions possibles.

La prise en compte, dans le cas général, de cette *sémantique des modèles stables* imposerait des raisonnements plus complexes que ceux que nous avons décrits dans ce chapitre : nous devrions générer, à la place d'une dérivation, un arbre de dérivations permettant de coder toutes les dérivations persistantes et complètes possibles. Il faudrait également définir la réponse à une requête comme une réponse dans *toutes* les dérivations persistantes et complètes. Heureusement, notre application n'a pas besoin d'une telle expressivité.

(ii) soit on impose des contraintes syntaxiques sur les ensembles de règles qui assurent l'existence d'une unique dérivation persistante et complète : on parle alors de *négation stratifiée*.

3.2.3.4 Négation stratifiée

Afin de répondre à ce problème, un langage comme Prolog impose un ordre d'application des règles : lorsqu'on a le choix entre deux applications de règles, celle qui a été donnée en premier dans le programme Prolog a la priorité. Pour donner une sémantique vraiment déclarative (c'est-à-dire indépendante de l'ordre des règles) à un ensemble de règles en présence de négation par l'échec (terme utilisé dans Prolog, nous considérons plutôt la négation stratifiée, que nous adaptons ici directement à nos $\mathcal{L}$-règles avec négation.

Définition 3.2.25 (Stratification) *Soit $\mathcal{R}$ un ensemble de règles. Une* stratification *de*

$\mathcal{R}$ est une partition ordonnée $(\mathcal{R}_1, \ldots, \mathcal{R}_k)$ de $\mathcal{R}$ (et pour $R \in \mathcal{R}_i$, on note $rank(R) = i$) telle que, pour tout prédicat (standard) p apparaissant dans la tête d'une règle de $\mathcal{R}$:

1. *si p apparait dans la partie positive d'une règle R' de $\mathcal{R}$, alors $rank(R) \leq rank(R')$;*
2. *si p apparait dans une partie négative d'une règle R' de $\mathcal{R}$, alors $rank(R) < rank(R')$.*

Un ensemble de règles est dit *stratifiable* si il admet une telle stratification. Tous les ensembles de règles ne sont pas stratifiables. Dans le premier exemple de la section 3.2.3.3, on doit avoir à la fois $rank(R_2) < rank(R_1)$ (car le prédicat B est dans la tête de R_2 et la partie négative de R_1) et $rank(R_1) \leq rank(R_2)$ (car le prédicat C est dans la tête de R_1 et la partie positive de R_2). Il n'y a donc pas de stratification possible. Dans l'exemple suivant, on doit à la fois avoir $rank(R_1) < rank(R_2)$ et $rank(R_2) > rank(R_1)$, ce qui est aussi impossible.

Exemple 3.2.11 *Soit l'ensemble de règles $\mathcal{R}$ composé des règles suivantes :*

$(R_1) : \forall X\ (R'_1(X) \wedge not\ R(X) \rightarrow S(X))$,
$(R_2) : \forall X\ (R'_2(X) \wedge not\ R(X) \rightarrow T(X))$,
$(R_3) : \forall X\ (R'_3(X) \wedge not\ T(X) \rightarrow U(X))$,
$(R_4) : \forall X\ (R'_4(X) \wedge not\ S(X) \wedge not\ U(X) \rightarrow V(X))$

On a les contraintes suivantes :

$rank(R_2) < rank(R_3) < rank(R_4)$ *et* $rank(R_1) < rank(R_4)$,

ce qui implique cinq possibilités :

1. $rank(R_1) < rank(R_2) < rank(R_3) < rank(R_4)$
2. $rank(R_2) < rank(R_1) < rank(R_3) < rank(R_4)$
3. $rank(R_2) < rank(R_3) < rank(R_1) < rank(R_4)$
4. $rank(R_1) = rank(R_2) < rank(R_3) < rank(R_4)$
5. $rank(R_2) < rank(R_1) = rank(R_3) < rank(R_4)$

correspondant respectivement aux 5 stratifications distinctes suivantes :

1. $\{R_1\}, \{R_2\}, \{R_3\}, \{R_4\}$
2. $\{R_2\}, \{R_1\}, \{R_3\}, \{R_4\}$
3. $\{R_2\}, \{R_3\}, \{R_1\}, \{R_4\}$
4. $\{R_1, R_2\}, \{R_3\}, \{R_4\}$
5. $\{R_2\}, \{R_1, R_3\}, \{R_4\}$

Une façon simple de savoir si un ensemble de règles $\mathcal{R}$ est stratifiable est de construire son *graphe de dépendance des prédicats intensionnels* $G_{\mathcal{R}}$. Ce graphe est construit de la façon suivante :

- les noeuds représentent les prédicats (standards) apparaissant dans les têtes des règles (appelés prédicats intensionnels)
- si p est un prédicat standard de la partie positive d'une règle et q un prédicat (standard) de sa tête, alors il existe un arc allant de p à q étiqueté **+** (dans notre représentation, un tel arc sera de couleur **verte**) ;
- si p est un prédicat standard de la partie négative d'une règle et q un prédicat (standard) de sa tête, alors il existe un arc allant de p à q étiqueté **–** (dans notre représentation, un tel arc sera de couleur **rouge**) ;

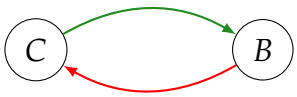
Dès lors, un ensemble de règles est stratifiable si et seulement si son graphe de dépendance des prédicats intensionnels ne contient aucun circuit négatif, c'est-à-dire, qu'aucun circuit du graphe ne contient d'arc négatif (arc de couleur **rouge**), voir exemples 3.2.12 et 3.2.13. On pourra alors facilement calculer la stratification à partir d'un parcours de ce graphe.

Notons que l'on peut donner des définitions plus générales de la stratification (qui considèrent comme stratifiables plus d'ensembles de règles), mais le critère défini ici (qui est celui de Datalog) suffit dans le cadre de notre application.

Exemple 3.2.12 *Soit l'ensemble des règles $\mathcal{R}'$ composé des règles suivantes :*

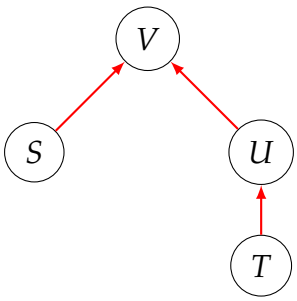
$(R_1) : \forall X\ (A(X) \wedge not\ B(X) \rightarrow C(X))$
$(R_2) : \forall X\ (C(X) \rightarrow B(X))$

Son graphe de dépendance des prédicats intensionnels sera représenté comme suit :



Le graphe contient un circuit avec un arc négatif, l'ensemble de règles $\mathcal{R}'$ n'est donc pas stratifiable.

Exemple 3.2.13 *Si nous reprenons l'exemple 3.2.11, son graphe de dépendance des prédicats est représenté ci-dessous.*



Nous voyons que ce graphe ne contient aucun circuit contenant un arc négatif, donc l'ensemble de règles $\mathcal{R}$ est stratifiable.

Une dérivation respecte une stratification lorsque, pour $1 \leq i < j \leq m$, toute application d'une règle de $\mathcal{R}_i$ précède toute application d'une règle de $\mathcal{R}_j$.

Dans la suite, nous ne nous intéressons qu'à des dérivations qui respectent la stratification. La propriété fondamentale est la suivante : *si un ensemble de règles est stratifiable, toute dérivation qui respecte la stratification est persistante, et pour une base de faits donnée, toutes les dérivations persistantes et complètes produisent le même résultat.*

3.3 Mappings

Les mappings sont des objets qui s'apparentent à des règles et sont composés d'une requête sur une première structure S_1 (le corps, écrit dans le langage de requête associé à S_1) et d'une requête d'insertion sur une autre structure S_2 (la tête, écrite dans le langage de requête associé à S_2).

Bien que le corps d'une requête puisse être écrit dans un langage assez expressif comme le langage SQL, nous devons également gérer, par exemple, les requêtes très simples sur des fichiers de tableur qui ne renvoient que des tuples correspondant aux lignes de la table. Dans ce cas, des contraintes peuvent être ajoutées au corps de la requête pour garantir qu'aucune donnée requise ne manque ou que les données satisfont à certaines conditions d'intégrité.

Pour pouvoir transformer les données issues d'une structure S_1 en données requises dans une structure S_2, nous nous appuyons également sur des fonctions qui apparaissent dans la tête d'un mapping.

Dans le chapitre 2, nous avions vu une forme simplifiée des mappings, que nous rappelons :

$$q_{S_1}(\overrightarrow{X}) \rightarrow q_{S_2}(\overrightarrow{X})$$

tel que q_{S_1} est une requête d'interrogation exprimée sur la structure S_1, dont les variables réponses sont $\overrightarrow{X}$, et q_{S_2} une requête d'insertion dans la structure S_2, qui insère les réponses retournées par q_{S_1}.

Cette forme est généralisée dans ce chapitre et adaptée à notre cas d'étude. Nos mappings ont la forme générale suivante :

$$Q_1(\overrightarrow{X}), C_1(\overrightarrow{X_1}), \ldots, C_k(\overrightarrow{X_k}) \rightarrow Q_2(f_1(\overrightarrow{Y_1}), \ldots, f_p(\overrightarrow{Y_p}))$$

où Q_1 est une requête sur la structure S_1 ; les réponses à Q_1 sont des substitutions des variables $\overrightarrow{X}$ par des valeurs dans S_1 ; C_i définissent des contraintes

($\overrightarrow{X_i} \subseteq \overrightarrow{X}$); et Q_2 est une requête d'insertion dans une structure S_2 telle que chaque $\overrightarrow{Y_i} \subseteq \overrightarrow{X}$ et chaque f_i est un symbole fonctionnel.

L'application d'un mapping m se fait comme suit : une réponse au corps de m est une substitution σqui est une réponse à Q_1 telle que $C_i(\sigma(\overrightarrow{X_i}))$ sont évalués à "vrai" pour tout $1 \leq i \leq k$. Étant donné une réponse σ au corps de m, chaque $f_i(\sigma(\overrightarrow{Y_i}))$ $(1 \leq i \leq p)$ est évalué, ce qui donne les valeurs v_i, et le tuple $(v_i)_{1 \leq i \leq p}$ est inséré à S_2 grâce à une requête d'insertion Q_2. Étant donné un ensemble de mappings à la structure S_2, la construction de S_2 est obtenue en effectuant toutes les applications de ces mappings.

Dans notre cas, nous utilisons 4 différents types de mappings :

- les *mappings de nettoyage* qui permettent, à partir d'un fichier texte formatté (comme csv), d'extraire et de nettoyer (grâce aux contraintes exprimées), de standardiser et d'insérer dans une base de données relationnelle les informations pertinentes en vue d'une utilisation automatisée des données. Les contraintes dans ces mappings sont utilisées pour sélectionner les lignes valides et les fonctions pour normaliser les valeurs ;
- les *mappings de données*, qui permettent le passage d'une base de données à une seconde base de données, en sélectionnant des valeurs de la première base de données, les agrégeant et les ajoutant à la deuxième base de données ; dans notre cas, cette seconde base de données, qu'on appellera *base de données de travail* ou *working database* intègre les diverses bases de données obtenues après nettoyage ;
- les *mappings data2knowledge*, qui vont d'une base de données (ici, la working database) à une base de faits : les informations pertinentes de la base de données sont transformées et traduites en faits. Comme les autres, cette étape sélectionne et agrège des valeurs mais surtout construit des faits de plus haut niveau. Une fois ces faits construits, des règles seront appliquées sur la base de faits ($\mathcal{F}$) jusqu'à saturation de celle ci ($\mathcal{F}^*$) ;
- les *mappings de stockage*, qui vont d'une base de faits vers une base de données relationnelle. Typiquement, une fois que notre base de fait est saturée, ces mappings sélectionnent la partie qui peut être interrogée par un utilisateur et l'importent dans une base de données relationnelle (ici, $MySQL$), ce qui permet de bénéficier de la puissance expressive de SQL pour l'interrogation. Ces mappings sont très simples dans notre cas puisqu'ils ne font que sélectionner des faits sans agrégation.

Ces mappings seront détaillés dans le chapitre 4, section 4.1.5.

CHAPITRE 4

Acquisition des données et connaissances

Dans le chapitre précédent (section 3.1), nous avons présenté l'architecture de notre système. La base de connaissances construite comporte une base de faits qui est alimentée, d'une part par des données issues de sources de données sur les traits fonctionnels des plantes, d'autre part par des connaissances provenant de la littérature scientifique et des experts en agroécologie. Dans ce chapitre nous détaillons la phase de sélection et de consolidation des données ainsi que la méthodologie d'acquisition et de formalisation des connaissances expertes, de façon à obtenir la base de faits initiale (c'est-à-dire avant application des règles qui produiront de nouveaux faits).

Le chapitre est structuré en deux sections :

1. La section 4.1 qui présente les différentes étapes ayant permis de construire des faits à partir des sources de données.
2. La section 4.2 qui explique le cheminement allant de l'acquisition des connaissances expertes à la construction de faits formalisant ces connaissances.

4.1 Des sources de données aux faits

Dans notre étude, nous nous intéressons à des sources de données qui fournissent des valeurs de traits fonctionnels pour des espèces de plantes, l'idée étant d'extraire ces valeurs de traits, puis de les consolider et transformer en connaissances sous forme de *faits* de la base de connaissances.

Jusqu'à présent, deux bases de données sont intégrées dans notre système : *TRY* [1] [Kat+20 ; Kat+11] qui sera considérée tout au long du chapitre comme la base de donnée principale et une autre base de données (confidentielle). Cette base de données, que nous appellerons *DB2*, a été construite dans le cadre d'une étude française sur les intérêts des cultures intermédiaires [2], et fournit des enregistrements pour 58 espèces selon 7 traits.

4.1.1 Présentation des sources de données utilisées

TRY - base de données mondiale sur les traits fonctionnels TRY est une base de données mondiale sur les traits fonctionnels des plantes, initiée par un réseau de chercheurs en écologie, sous la direction de Future Earth et de l'Institut Max Planck de biogéochimie. Elle a vocation à intégrer, harmoniser et rendre accessibles les données issues de nombreuses bases collectives déjà existantes, mais également tout ensemble de données publié ou non publié. Créée en 2007, elle contient maintenant près de 11 millions d'enregistrements et des informations sur 2100 traits concernant 160 000 taxons (les taxons étant des espèces, genres ou familles), dont la grande majorité sont des espèces. Actuellement, TRY intègre, dans une même structure, des données de plus de 400 sources en écologie, parmi lesquelles on peut citer *LEDA* [Kne+03] (sur la flore européenne), *BiolFlor* [KDK04] (sur la flore d'Allemagne), *BROT* [TP18] (sur la flore méditerranéenne) ou *FRED* [Ive+17] (spécialisée dans les traits racinaires). Comme expliqué par la suite, la base de données TRY n'est pas directement accessible : il faut effectuer une demande comportant les traits d'intérêt, puis les données sont fournies sous la forme d'un fichier texte formaté. Nous avons demandé 52 traits, ce qui nous a donné 151k espèces.

D'une manière générale, les données fournies par TRY comprennent des valeurs de traits fonctionnels pour des espèces et des informations sur le contexte dans lequel ont été mesurés les traits (comme la géolocalisation, certaines caractéristiques du sol : pH, teneur en azote, teneur en phosphore, le stade phénologique des espèces, etc). Dans ce chapitre, nous nous concentrerons sur les valeurs de traits fonctionnels. Les seules informations contextuelles prises en compte seront les conditions de culture des plantes, car nous avons constaté qu'elles avaient une grande incidence sur la variabilité des valeurs de traits au sein d'une même espèce. Nous n'avons pas pris en compte les autres informations contextuelles, d'abord parce qu'elles ne sont pas suffisamment renseignées, et ensuite, parce

1. www.try-db.org
2. https://methode-merci.fr

Trait	ObsNum	ObsGRNum	AccSpecNum	Definition
Bark calcium (Ca) content per bark dry mass	30	30	5	Under Construction
Bark carbon (C) content per bark dry mass	772	534	275	Definition
Bark carbon (C) isotope signature (delta 13C)	13	13	1	Under Construction
Bark carbon/nitrogen (C/N) ratio	13	13	1	Under Construction
Bark copper (Cu) content per bark dry mass	30	30	5	Under Construction
Bark crystals	3593	2747	1644	Under Construction
Bark density (bark dry mass per bark volume)	737	737	274	Definition

FIGURE 4.1 – Correspondances entre les traits de TRY et de TOP. Ce tableau correspond à un extrait d'un tableau de TRY plus grand contenant ces informations pour tous les traits : https://www.try-db.org/de/TabDetails.php.

lorsqu'elles le sont, elles ont généralement des valeurs non standardisées qui nécessitent un travail de nettoyage avant de pouvoir être exploitées automatiquement. Nous verrons dans la suite du chapitre que ces problèmes (valeurs manquantes, hétérogénéité des valeurs) se posent déjà pour l'exploitation des valeurs de traits fonctionnels.

TRY harmonise les noms des espèces. En effet, tous les noms des taxons utilisés dans TRY sont consolidés par rapport à *The Plant List* [Kal12], en utilisant le *Taxonomic Name Resolution Service* [Boy+13]. Le nom du taxon une fois consolidé est reporté dans la colonne *AccSpeciesName* et est appelé *nom accepté*.

Concernant les noms de traits, TRY fait des correspondances entre ses noms de traits et ceux du *Thesaurus Of Plant characteristics* (TOP, [Gar+17]). Cette étape de correspondance n'est pas encore complètement achevée, en effet, seulement environ 15% des traits ont une correspondance dans TOP parmi tous les traits de TRY. En particulier, nous avons observé que 8 de nos 52 traits ont une correspondance avec TOP.

La figure 4.1 présente un tableau dans lequel sont présents quelques traits de

TRY (le nom des traits est dans la colonne *Trait*). La correspondance avec TOP se fait dans la dernière colonne, *Definition*. On remarque que deux valeurs sont possibles dans cette colonne : (a) soit la valeur *definition*, lorsque le trait TRY a été identifié avec un trait TOP, comme le cas, dans la figure 4.1 des traits *Bark carbon (C) content per bark dry mass* et *Bark density (bark dry mass per bark volume)* ; (b) soit la valeur *under construction*, dans le cas où l'identification n'a pas encore été faite, comme pour tous les autres traits de la table.
Les autres colonnes nous donnent des précisions, pour chaque trait, sur le nombre d'observations (colonne *ObsNum*), le nombre d'observations géo-référenciées (colonne *ObsGRNum*) et le nombre d'espèces (colonne *AccSpecNum*).

DB2 - base de données sur les traits fonctionnels développée dans le cadre d'un projet national sur les traits et services des cultures intermédiaires La seconde base de données, DB2, contient également des valeurs de traits fonctionnels mais est de très petite taille. En effet, elle donne des informations sur 7 traits fonctionnels pour 58 espèces. Cette base de données, comme beaucoup en agronomie, a été construite afin d'être utilisée pour un cas d'étude spécifique [3], d'où le fait qu'elle ne regroupe que très peu de mesures de traits pour une liste d'espèces d'intérêt pour l'étude en question. À la différence de TRY, ici les valeurs sont standardisées, mais aucun élément de contexte n'y est mentionné.

Bien que cette base ne contienne que très peu d'informations, elle est pertinente pour nous car elle contient des espèces qui nous intéressent (des herbacées présentes dans les écosystèmes cultivés) pour lesquelles certains traits n'ont pas de valeur dans TRY. Notons que, comme dans TRY, les noms des espèces ont été normalisés en utilisant *The Plant List*.

4.1.2 Structuration des données dans les bases

Le cas de TRY Le fichier de données reçu de TRY est constitué d'un tableau contenant 27 colonnes (voir figure 4.2 pour plus d'informations sur chaque colonne).

Dans TRY, les données sont structurées en observations. Une observation (associée dans TRY à un identifiant unique, colonne *ObservationID*) concerne des mesures de traits (colonne Trait) muni de leurs IDs (TraitID) ainsi qu'une plante d'une certaine espèce. Cependant, on n'a pas une unique valeur mesurée pour le trait lui-même, mais plusieurs valeurs, chacune associée à un "nom de données"

3. Ici, en association avec le développement de la méthode MERCI pour estimer l'intérêt de cultures intermédiaires, https://agriculture-de-conservation.com/sites/agriculture-de-conservation.com/IMG/pdf/tcs112_culture3.pdf

Nom de colonne	**Commentaires**
LastName	Nom du contributeur
FirstName	Prénom du contributeur
Dataset	Nom de la source de données
DatasetID	Identifiant unique de la source de données
SpeciesName	Nom de l'espèce
AccSpeciesName	Nom de l'espèce consolidée
AccSpeciesID	Identifiant unique de l'espèce consolidée
ObservationID	Identifiant unique de l'observation
ObsDataID	Identifiant unique pour chaque enregistrement
TraitName	Nom du trait
TraitID	Identifiant unique du trait
OriginalName	Nom du sous-trait ou de l'information de contexte
DataName	Nom consolidé du sous-trait ou information de contexte
DataID	Identifiant unique du sous-trait ou de l'information de contexte
OrigValueStr	Valeur brute du trait
OrigUnitStr	Unité de la valeur brute
ValueKindName	Type de valeur (mesure unique, moyenne de mesure, etc)
OrigUncertaintyStr	Incertitude
UncertaintyName	Incertitude consolidée
Replicates	Nombre de répétitions
StdValue	Valeur standardisée
StdUnit	Unité de la valeur standardisée
RelUncertaintyPercent	Incertitude en pourcentage
OrigObsDataID	Identifiant unique pour les doublons
ErrorRisk	Indication sur la fiabilité de la valeur
Reference	Référence à citer si l'enregistrement du trait est utilisé
Comment	Explications sur la valeur brute

FIGURE 4.2 – Informations disponibles dans TRY

(DataName), qui correspond à une spécification plus précise du trait. Une observation est décrite par plusieurs lignes, qui fournissent des mesures de ce trait ainsi que leurs unités de mesure, et les relient à des informations contextuelles (appelées *covariates*). Chaque observation fournit également le nom et l'identifiant de la source de données qui a produit les mesures d'origine (respectivement en colonnes *Dataset* et *DatasetID*).

Plus précisément, toutes les informations de traits et de contextes sont :

(i) Dans un premier temps associées à un nom de données, *DataName* accompagné d'un identifiant unique *DataID* ; leurs valeurs brutes ainsi que l'unité de mesure associée sont présentes dans les colonnes *OrigValueStr* et *OrigUnitStr*, respectivement. Notons que dans la colonne OrigValueStr, les valeurs ne sont pas standardisées (on peut avoir des valeurs de types différents, voir section 4.1.4.1) et des unités différentes. Cependant, pour les traits quantitatifs ayant plus de 1000 enregistrements, TRY a effectué des standardisations des unités de mesure : en effet, les valeurs brutes récupérées des différentes bases ayant des unités différentes ont été converties en des valeurs ayant une même unité ; ces valeurs dites standardisées, sont présentes dans la colonne *StdValue* et leur unité dans la colonne *StdUnit*. Pour ces valeurs standardisées, TRY a fait certains contrôles de qualité des données, en particulier une estimation de la (non-)fiabilité des valeurs appelée risque d'erreur est donnée (la valeur du risque d'erreur est présente dans la colonne *ErrorRisk*).

(ii) Dans un second temps, s'il s'agit d'informations de traits, les DataNames sont regroupés pour créer un nom de trait, dans la colonne *TraitName* ainsi qu'un ID, colonne *TraitID*. S'il s'agit d'informations de contexte, les colonnes TraitID et TraitName n'ont pas de valeur (on retrouvera une cellule vide dans le tableau). Notons que très souvent, plusieurs DataID sont rassemblés en un même TraitID.

Prenons l'exemple du trait *Fine root length per fine root dry mass (specific fine root length, SRL)* qui a pour ID (traitID) 614. Comme montré dans la figure 4.3, il est constitué de 7 sous-groupes (DataNames). On voit que les DataNames apportent des précisions sur la nature du trait, en faisant la distinction entre des enregistrements réalisés sur des racines très fines, fines ou moyennes. L'importance de faire cette distinction entre DataName et TraitName apparaît lorsqu'on souhaite normaliser les valeurs de traits (voir la section 4.1.5).

Pour les deux derniers sous-groupes (DataIDs 5155 et 5156) de la figure 4.3, les valeurs des racines fines retenues sont les valeurs minimum et maximum des mesures. Notons que ce critère lié aux valeurs retenues, est indiqué dans la colonne

DataID	DataName
1359	Fine Roots Specific Root Length (srl, length/dry mass)
4038	Fine root mass :length ratio
4201	Very fine root (0-1mm) specific root length (srl)
4202	Medium fine root (1-2mm) specific root length (srl)
5154	Fine root specific length (fine root length per dry mass; diameter not specified)
5155	Fine root specific length (fine root length per dry mass; diameter not specified) : minimum
5156	Fine root specific length (fine root length per dry mass; diameter not specified) : maximum

FIGURE 4.3 – DataNames du traitID 614

ValueKindName de TRY, avec une valeur standardisée parmi celles présentes dans la figure 4.4.

Dans cette figure (figure 4.4), nous indiquons les valeurs possibles que peut prendre la colonne *ValueKindName* (colonne *Type*), une précision sur chaque valeur (colonne *Commentaire*) ainsi que la proportion de chaque type dans TRY (colonne *Pourcentage*). Dans notre cas, nous ne considérons que les mesures de type *single*, *mean*, *best estimate* et celles dont le type n'est pas précisé (*NULL*) car ces dernières constituent la plus grande proportion des mesures. Les autres mesures ne sont pas prises en compte dans les calculs (de valeurs, fonctions et services).

Le cas de DB2 DB2 est également constituée d'un tableau dont chaque ligne représente une espèce, et les colonnes sont les traits fonctionnels. Les lignes sont donc indépendantes les unes des autres. Toutes les valeurs présentes sont quantitatives.

4.1.3 Accès aux données

Le cas de TRY L'accès aux données de TRY se fait par une demande dans laquelle l'utilisateur choisit les traits qui l'intéressent et possiblement une liste d'espèces. Une fois la demande faite, l'utilisateur reçoit les données publiques ainsi que les données restreintes ayant reçu l'accord des contributeurs, dans un fichier texte formaté, d'une façon similaire à csv. Ce dernier contient les enregistrements des traits demandés, c'est-à-dire les valeurs de traits pour les espèces, ainsi que les informations contextuelles les accompagnant.

Pour notre cas d'étude, nous avons fait la demande de 52 traits, que les ex-

Type	Commentaire	Pourcentage
Single	mesure unique	26%
Mean	moyenne de plusieurs mesures	1%
Best estimate	estimation de la mesure faite par TRY	< 1%
Maximum	valeur maximum	< 1%
Minimum	valeur minimum	< 1%
Median	valeur médiane	< 1%
Upper quartile	3ème quartile (valeur qui sépare les 75% inférieurs des données)	< 1%
Lower quartile	1er quartile (valeur qui sépare les 25% inférieurs des données)	< 1%
Modal	valeur la plus fréquente (mode)	< 1%
NULL	pas d'information sur le type considéré	72%

FIGURE 4.4 – Types de mesure présents dans TRY(la colonne Type du tableau correspondant à la colonne ValueKindName dans TRY). Puis la proportion de chaque type de mesure parmi toutes les mesures dans TRY (colonne Pourcentage).

perts ont jugé utiles pour la correspondance avec les traits des diagrammes trait-fonction-service (voir la section 4.2.2). Par contre, nous ne nous sommes pas restreints à une liste d'espèces en particulier. En effet, bien que notre étude concerne uniquement les herbacées, le formulaire de demande à TRY ne comportait pas de filtre permettant de ne sélectionner que les herbacées, et nous ne voulions pas nous restreindre à une liste d'espèces a priori.

Pour cette demande [4], nous avons reçu un peu plus de 24.3M d'enregistrements parmi lesquels nous avons 1.6M d'observations. Ces observations concernent 150976 taxons (parmi les 160000 au total dans TRY, soit $\approx$ 94% de taxons de TRY) : autrement dit, chaque taxon a au moins 1 valeur pour au moins 1 trait (parmi les 52 retenus). Parmi ces taxons, il y a 70152 herbacées (soit $\approx$ 46% des taxons).

Le cas de DB2 DB2 n'est pas disponible de façon publique. Pour y avoir accès, nous avons contacté les responsables du projet qui ont accepté de nous communiquer cette base. Les données nous ont été fournies dans un fichier sous format csv (similairement à TRY).

TraitID	TraitName	Pourcentage de taxons	Pourcentage d'herbacées
1021	Plant carbon/nitrogen (C/N) ratio	0.01	100
409	Shoot carbon/nitrogen (C/N) ratio	1.36	60
146	Leaf carbon/nitrogen (C/N) ratio	3.12	39
150	Litter carbon/nitrogen (C/N) ratio	0.1	21
77	Plant growth rate relative (plant relative growth rate, RGR)	0.56	61
8	Plant nitrogen(N) fixation capacity	8.89	34
700	Plant biomass and allometry : Plant dry mass	0.15	83
388	Plant biomass and allometry : Leaf dry mass (paracotyledons) per plant	0.04	21
403	Plant biomass and allometry : Shoot dry mass (plant aboveground dry mass) per plant	0.5	90
1508	Root length per soil volume	0.02	42
2025	Fine root length per soil volume	0.03	69
2281	Fine root (absorptive) length per soil volume	0.02	4
3086	Leaf area per leaf dry mass (specific leaf area, SLA or 1/LMA) petiole, rhachis and midrib excluded	0.27	22
3117	Leaf area per leaf dry mass (specific leaf area, SLA or 1/LMA) : undefined if petiole is in- or excluded	8.3	36
3116	Leaf area per leaf dry mass (specific leaf area, SLA or 1/LMA) : petiole included	4.4	53
3115	Leaf area per leaf dry mass (specific leaf area, SLA or 1/LMA) : petiole excluded	4.9	44
47	Leaf dry mass per leaf fresh mass (leaf dry matter content, LDMC)	4.3	59
1080	Root length per root dry mass (specific root length, SRL)	0.33	81
614	Fine root length per fine root dry mass (specific fine root length, SRL)	0.7	56
339	Shoot nitrogen (N) content per shoot dry mass	0.13	97
408	Plant biomass and allometry : Shoot nitrogen (N) content per plant	0.08	98
502	Plant biomass and allometry : Leaf nitrogen (N) content per plant	0.08	50
1126	Shoot organic nitrogen (N) content per shoot dry mass	0.08	80

FIGURE 4.5 – Quelques traits de TRY avec leurs IDs. Pour chaque trait, nous avons le pourcentage de taxons (parmi les 150976) et le pourcentage d'herbacées (par rapport au pourcentage de taxons). Voir l'annexe A.1 pour les informations relatives aux 52 traits.

4.1.4 Difficultés rencontrées liées aux données

Dans cette section, nous discutons des deux difficultés majeures que nous avons rencontrées pour l'exploitation des données, notamment celles de TRY : problème des valeurs non standardisées et problème des valeurs manquantes. Chaque difficulté est intervenue à un stade différent de la construction du système. La première difficulté est apparue lors de l'utilisation des données, puisqu'on s'est aperçu que malgré l'effort de standardisation des valeurs de TRY, il y avait toujours une forte hétérogénéité dans les valeurs.

La seconde difficulté a surgi après la saturation de la base de faits, lors du requêtage. En effet, à ce moment là, nous avons remarqué que peu d'espèces remontaient jusqu'aux services (c'est-à-dire que peu d'espèces avaient finalement une valeur de service), la raison étant qu'il y a une quantité importante de valeurs manquantes. Une analyse de ce majeur problème des valeurs manquantes est présente dans [Ka20]. Dans la suite, nous détaillons ces deux difficultés.

4.1.4.1 Problème des valeurs non standardisées

De façon générale dans les données reçues de TRY, les cellules ont un contenu hétérogène en termes d'unités de mesure, de valeurs prises par un champ non numérique et de type d'information contextuelle.

Parmi nos 52 traits, seuls 8 ont toutes leurs valeurs exprimées dans une unité de mesure standardisée. Pour les traits quantitatifs n'ayant pas une unité de mesure standardisée, nous avons décidé de ne pas faire de conversion mais plutôt de garder l'unité la plus fréquente. C'est-à-dire que les valeurs qui ne sont pas dans l'unité la plus fréquente ont été écartées. Notons que cette étape n'écarte que très peu de valeurs pour chaque trait. Et, de façon plus générale, toutes les étapes de nettoyage n'enlèvent que très peu de valeurs d'espèces (voir figure 4.7).

Pour les traits non quantitatifs, les valeurs sont très diverses. Par exemple, le trait Plant Life Span (relatif à la pérennité de la plante) prend des valeurs de type chaîne de caractère parmi ["Bisannual", "Annual", "Biennial", "Perennial"] mais on trouve aussi beaucoup d'autres valeurs comme "perennial < 20 years", "biasannual", "pere", "nope", "from few decades to more than 60 years", "1", "2", "3", "winter annual", "shrub", "woody", etc.

Par conséquent, une étape de nettoyage est obligatoire avant que cette riche source d'information puisse être exploitée de manière automatisée. Cette étape, principalement basée sur la recherche de chaînes de caractères, élimine les infor-

4. La demande a été effectuée en juillet 2021.

mations non pertinentes, douteuses (selon le risque d'erreur indiqué lorsqu'il est disponible) ou inutilisables, et transforme les valeurs des champs conservés. Elle est faite à l'aide de mappings et sera décrite dans la section 4.1.5.

4.1.4.2 Problème des valeurs manquantes

Bien que les herbacées soient bien représentées, c'est-à-dire que parmi les espèces ayant au moins une mesure pour au moins un trait dans TRY, un grand nombre sont des herbacées (voir figure 4.5 pour plus de précision sur les quantités d'herbacées parmi les taxons) et que les 52 traits recouvrent la quasi totalité des taxons disponibles dans TRY, nous avons constaté que beaucoup de valeurs de trait manquent. En guise d'illustration, la figure 4.5 contient un sous-ensemble des 52 traits (ceux qui interviennent dans le rendu du service de fourniture d'azote à la vigne). Les deux premières colonnes correspondent respectivement aux IDs et noms de ces traits dans TRY. La 3ème colonne, *Pourcentage de taxons*, nous donne la proportion d'espèces (parmi les 150976 taxons reçus) qui ont au moins une mesure pour ce trait. Par exemple, si on considère le trait ID 409 (ayant pour nom "Shoot carbon/nitrogen (C/N) ratio" dans TRY), 1.36 % des taxons, parmi 150976, ont une mesure. Enfin, la quatrième colonne, *Pourcentage d'herbacées* correspond à la proportion d'herbacées parmi les taxons ayant une mesure. Le trait ID 409 est renseigné pour 1.36 % des taxons (colonne 3) parmi lesquels 60% (colonne 4) sont des herbacées.

La figure A.1 en annexe présente un tableau plus complet avec ces informations sur les 52 traits d'intérêt. Nous pourrons remarquer que de façon générale nos traits ne recouvrent pas beaucoup d'espèces. En effet, à part le trait ayant pour nom *Plant growth form* (et pour ID 42) dans TRY, qui recouvre $\approx$ 97% des taxons parmi lesquels $\approx$ 98% sont des herbacées, les autres traits couvrent au mieux $\approx$ 11% des taxons (pour le trait TRY *Plant life form (Raunkiaer life form)*, ID 343).

D'où le problème majeur de valeurs manquantes dans TRY. Pour pallier ce problème, nous avons mis en place plusieurs stratégies : d'abord nous avons décidé de quantifier ce manque de valeurs afin de le signaler à l'utilisateur ; ensuite, nous avons ajouté d'autres bases de données que TRY à notre système ; et enfin, nous avons mis en place un processus de prise en compte des retours utilisateurs.

1. Concernant le premier point, la quantification du manque s'est faite par l'ajout d'un paramètre, appelé *fiabilité*. Ce paramètre indique la proportion de traits présents (parmi ceux nécessaires) qui nous ont permis de calculer une valeur de service. Par exemple, si un service est relié à 4 traits (par

l'intermédiaire des fonctions écosystémiques) et que seuls 2 traits parmi les 4 ont une mesure pour une espèce donnée, alors, la valeur calculée de service a une fiabilité de 1/2, indiquant que la valeur de service pour cette espèce a été calculée avec une proportion de 50% des traits. Cette partie est gérée par des règles en utilisant des fonctions calculées. Elle sera détaillée dans la section 5.1.5 du prochain chapitre.

2. Par rapport au deuxième point, nous avons ajouté une autre base de données (DB2), qui a l'avantage de contenir des informations complémentaires à celles de TRY. La difficulté ici a été de trouver d'autres bases de données pertinentes. En effet, TRY est une base de données mondiale qui intègre déjà les principales sources de données existantes en écologie sur les traits fonctionnels. La production de données sur les traits fonctionnels en agronomie commence tout juste à se développer.
3. Pour le dernier point (la prise en compte des retours utilisateurs), l'idée est de permettre à l'utilisateur d'interagir avec l'outil au niveau des traits, des fonctions et des services. En effet, un mécanisme a été mis en place pour permettre à l'utilisateur de compléter, affiner ou contredire des valeurs données par l'outil. Faire entrer l'utilisateur dans la boucle permettra non seulement d'ajuster les valeurs des traits, fonctions ou services mais aussi de combler, éventuellement, certaines valeurs manquantes. Cette étape est également gérée par les règles et sera par conséquent détaillée dans le chapitre 5 en section 5.1.5.

4.1.5 Mappings : passage des données aux faits

Le passage des données brutes aux faits s'est fait en plusieurs étapes, utilisant à chacune un type différent de mappings.

Mappings de nettoyage : des fichiers aux bases de données Comme nous l'avons expliqué dans la section 4.1.4.1, une série de nettoyages a dû être faite sur les données (de TRY) avant de pouvoir les exploiter. Cette étape présentée en figure 4.6 correspond à la première étape de l'architecture de l'outil présentée en figure 3.1 :

Parmi ces nettoyages nous pouvons citer :

- les nettoyages qui restreignent les valeurs retenues à stocker dans la base de données. En effet, nous avons décidé de ne garder que des valeurs de type (dont le ValueKindName) soit égal à single, mean ou best estimate.

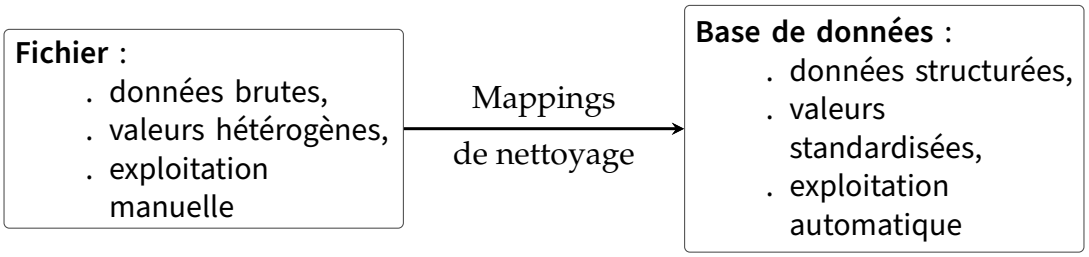


FIGURE 4.6 – Passage des fichiers aux bases de données : mappings de nettoyage

Même si TRY mène un effort de standardisation des valeurs (colonne StdValue), ce n'est pas encore le cas pour la majorité des traits que nous avons demandés, nous avons donc décidé de récupérer la valeur en colonne StdValue si elle est présente, sinon, de prendre la valeur originelle dans la colonne OrigValueStr à condition que celle-ci ait une unité standardisée [5] ; enfin pour les valeurs standardisées par TRY (c'est-à-dire celles qui sont dans la colonne StdValue), une recommandation de TRY alerte qu'une donnée ayant un risque d'erreur trop grand (ErrorRisk > 4) indique une anomalie dans les données, nous avons donc décidé de retirer ces valeurs de notre étude ;

- les nettoyages qui catégorisent les mesures. En effet, nous avons regroupé les mesures dans TRY suivant 2 types de conditions (qu'on appelle *growing conditions*) : les conditions *naturelles*, c'est-à-dire les mesures qui ont été prises dans des conditions peu contrôlées, comme ["Field", "Common Garden", "Natural Environment", "Botanical garden", "outside natural vegetation", "park", "plantation", etc] ; et les conditions *expérimentales* c'est-à-dire les mesures prises dans des conditions très contrôlées, comme ["Growth Chamber", "Greenhouse :highlight_highpH_competition", "Controlled climate chamber", "experimental treatment", "Glass house", etc]. Cette catégorisation s'est faite à l'aide des agronomes, avec qui nous avons exploré toutes les valeurs rencontrées du sous-trait *Exposition* ayant pour ID 327 dans TRY, et regroupé ces valeurs en deux grandes valeurs possibles. Par la suite, un attribut a été ajouté pour mentionner le type de conditions : les valeurs possibles de cet attribut sont *n* (pour conditions naturelles), *e* (pour conditions expérimentales) et *NULL* si les conditions ne sont pas précisées. Ajoutons également qu'il sera possible dans les faits de combiner ces trois valeurs en une valeur *t* (pour conditions totales), qui regroupe toutes les

5. Notons que l'unité standardisée d'un trait est soit l'unité des valeurs standardisées (si le trait a été standardisé par TRY) sinon c'est l'unité la plus fréquente dans les valeurs originelles. Nous n'avons fait aucune conversion de valeurs.

mesures quelles que soient leurs conditions (naturelles, expérimentales et non précisées).

- et les nettoyages qui transforment des valeurs pour construire de nouveaux attributs. En effet, puisqu'on s'intéresse aux herbacées dans notre cas d'étude, une étape de reconnaissance des herbacées parmi les espèces présentes dans les bases de données est nécessaire. Pour cela, on construit un attribut booléen disant si une espèce est une herbacée ou non, ce qui permet de stocker toutes les espèces dans la BD, mais en sachant si ce sont ou non des herbacées. Dans TRY, la reconnaissance des herbacées se fait en croisant les valeurs de 2 traits fonctionnels : *Plant Growth Form* ayant pour ID 42 et *Plant Life Form* ayant pour ID 343. C'est-à-dire, on sélectionne un sous-ensemble, S_1, contenant des valeurs parmi les valeurs du trait ID 42 et un sous-ensemble de valeurs S_2, contenant des valeurs parmi celles du trait ID 343.

  S_1 est composé des valeurs brutes du trait ID 42 suivantes : ["herb","cereal", "forb","forbs","graminoid","graminoids","graminoids Tussock","geophyte", "herbaceous legume", "Forb/herb", "grass", "forage grass", "pasture grass", "praire grass", "weed", "weed,sedge", "crop", "crops", "Small_Herb_", "Perennial Herb/Hemicryptophyte", "Terrestrial_Herb", "Herb_Erect", Terrestrial(?) Herb", "Herbaceous/Terrestrial Herb", "Annual Herb", "Perennial Grass/Hemicryptophyte", "perennial herb", "perennial graminoid", "perennial leguminous herb", "graminoid/fern"]

  S_2 est composé des valeurs brutes du trait ID 343 suivantes : ["Geophyte", "Hemicryptophyte", "Therophyte", "always Therophyte", "always Geophyte", "always Hemicryptophyte", "always Hemicryptophyte, always Therophyte" "always Geophyte, always Hemicryptophyte" "questionable Hemicryptophyte, always Therophyte" "always Hemicryptophyte, sometimes Therophyte" "sometimes Geophyte, always Hemicryptophyte" "sometimes Hemicryptophyte, always Therophyte" "always Geophyte, always Therophyte" "always Geophyte, sometimes Therophyte"].

  Une espèce ayant une valeur du trait ID 42 incluse dans S_1 ou une valeur du trait ID 343 incluse dans S_2 est considérée comme une herbacée.

Plus formellement, la forme générale des mappings de nettoyage est la suivante

$$q_{\text{csv}} \rightarrow q_{\text{DB}}$$

où

$q_{\text{csv}} = \text{SELECT } \overrightarrow{X} \text{ FROM line}(\overrightarrow{Y}), \text{valid}_1(\overrightarrow{Y_1}), \ldots, \text{valid}_k(\overrightarrow{Y_k}),$
$q_{\text{DB}} = \text{INSERT } (f_1(\overrightarrow{X_1})), \ldots, f_p(\overrightarrow{X_p})),$
$\overrightarrow{X}, \overrightarrow{Y_1}, \ldots, \overrightarrow{Y_k}$ sont des sous-listes de $\overrightarrow{Y}$ et $\overrightarrow{X_1}, \ldots, \overrightarrow{X_p}$ des sous-listes de $\overrightarrow{X}$
et $f_1, \ldots, f_p$ sont des fonctions calculées.

La requête q_{csv} permet de sélectionner et lire chaque ligne du fichier csv, où $\overrightarrow{Y}$ est la liste des valeurs de la ligne, et de vérifier si la ligne satisfait tous les tests de validité exprimés dans les prédicats $valid_1(\overrightarrow{Y_1}), \ldots, valid_k(\overrightarrow{Y_k})$, où chaque $\overrightarrow{Y_i}$ est une sous-liste de $\overrightarrow{Y}$. Si le contrôle de validité réussit, la sous-liste de valeurs $\overrightarrow{X}$ est passée à la requête q_{DB}, qui insert une liste de p valeurs dans la base de données DB, chacune de ces valeurs étant construite à partir d'une sous-liste $\overrightarrow{X_i}$ de $\overrightarrow{X}$ en utilisant une fonction de transformation f_i.

À la fin de cette étape, chaque tuple *(ID de l'observation, ID de l'espèce, ID du trait, ID du sous-groupe, Condition de croissance)* apparaissant dans la base de données obtenue a une seule valeur de dataName exprimée dans une unité standardisée (c'est-à-dire que nous avons choisi une unité de mesure ou une liste de valeurs possibles pour chaque dataName).

Bien que le nettoyage aurait pu porter sur l'ensemble des données reçues de TRY et être indépendant du cas d'utilisation spécifique de l'enherbement des vignes, nous avons effectué un nettoyage sélectif pour des raisons de temps. En particulier, seules les espèces herbacées sont pertinentes pour le cas d'utilisation ; nous avons construit cette catégorie (qui n'est pas une catégorie bien définie en ce qui concerne la taxonomie des plantes) à partir de valeurs spécifiques des IDs de traits, afin de distinguer les espèces herbacées des autres. D'autres catégorisations pourraient être faites pour d'autres cas d'utilisation.

Finalement, la base de données TRY est composée d'une seule table comportant 29 attributs, tels que 27 d'entre eux correspondent aux colonnes de TRY présentes dans la figure 4.2 et les deux autres, *MyStdValue* et *GrowingCondition* représentent respectivement les valeurs retenues, standardisées et les conditions de croissance.

Notons que la source de données DB2 considère uniquement des herbacées avec des conditions de croissance naturelles et n'a donc pas nécessité de nettoyage.

Remarquons que les nettoyages effectués ne sont pas la cause des valeurs manquantes dans TRY. En effet, la figure 4.7 nous donne pour quelques traits de TRY (les Trait ID sont dans la 1ere colonne), le nombre d'espèces avant les nettoyages (colonne 2) et celui après tous les nettoyages (colonne 3). La proportion

d'espèces gardée après les nettoyages est donné en colonne 4. Nous constatons que pour la plupart des traits, nous ne perdons que très peu d'espèces, sauf pour le trait ID 409, pour lequel 1897 espèces sont éliminées (nous avions 1951 espèces avant les nettoyages et passons à 54 espèces après les nettoyages). Cependant, le trait ID 409 est interchangeable avec d'autres traits, notamment les traits IDs 150 et 409, pour lesquels les nettoyages éliminent peu d'espèces.

Mappings de bases de données : des bases de données à la *working database* Pour construire la base de données de travail, nous avons sélectionné les espèces herbacées de notre base de données locale TRY (ce qui correspond à environ la moitié des espèces de TRY) et avons agrégé toutes les valeurs de dataName (ou sous-groupe) provenant de différentes observations pour un tuple donné (ID de l'espèce, ID du trait, ID du sous-groupe, Condition de croissance). L'agrégation est effectuée en prenant la moyenne des valeurs. Cette étape (voir figure 4.8) correspond à la deuxième étape de l'architecture en figure 3.1.

DB2 est simplement importée dans la base de données de travail. La forme générale d'un mapping de base de données est

$$q_{\mathrm{DB}_i} \rightarrow q_{wDB}$$

où q_{DB_i} est une requête de sélection sur la base de données DB_i et q_{wDB} insère les tuples retournés par q_{DB_i} dans la base de données de travail (wDB).

À la fin de cette étape, la base de données de travail contient une seule valeur de sous-groupe pour chaque tuple *(ID de l'espèce, ID du trait, ID du sous-groupe, Condition de croissance, Base de données source)* et ne contient que des espèces herbacées. Notons que, à la différence de TRY, dans le cas de la base DB2, il n'y a pas de distinction faite entre un sous-groupe et un trait. On suppose donc que ces deux attributs sont les mêmes et auront les mêmes valeurs.

Finalement, la base de données de travail est composée des attributs : Nom de l'espèce, ID de l'espèce, ID du trait, ID du sous-groupe, Valeur (moyenne), Condition de croissance, Nom de la base de données source.

Mappings data2knowledge : de la working database à la base de faits La dernière étape (voir figure 4.9) nous permet de construire les faits à partir de la working database. Cela correspond au passage du niveau données au niveau conceptuel présenté dans la figure 3.1 :

Trait ID	#espèces avant nettoyages	#espèces après nettoyages	Proportion
3086	341	341	100%
1126	115	115	100%
1508	24	24	100%
2013	97	97	100%
502	2	2	100%
2025	51	51	100%
339	196	196	100%
86	99	99	100%
88	122	122	100%
116	17	17	100%
47	5769	5696	99%
3115	7341	7269	99%
82	404	398	98%
21	2397	2360	98%
1080	458	449	98%
614	1018	997	98%
403	681	671	98%
3116	6813	6701	98%
3117	12178	12001	98%
77	844	818	97%
700	163	159	97%
2281	27	26	96%
83	403	388	96%
410	595	563	95%
146	4351	4009	92%
8	13010	11670	90%
388	68	48	71%
150	156	90	58%
409	1951	54	3%

FIGURE 4.7 – Nombre d'espèces avant et après les nettoyages pour quelques traits.

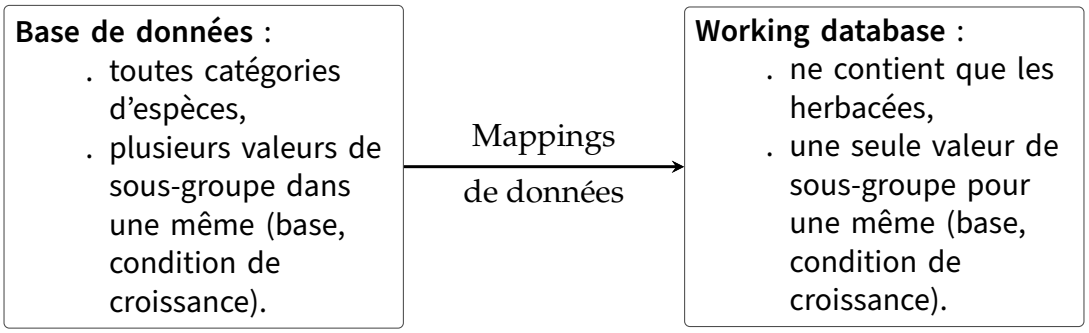


FIGURE 4.8 – Passage des bases de données à la working database : mappings de données

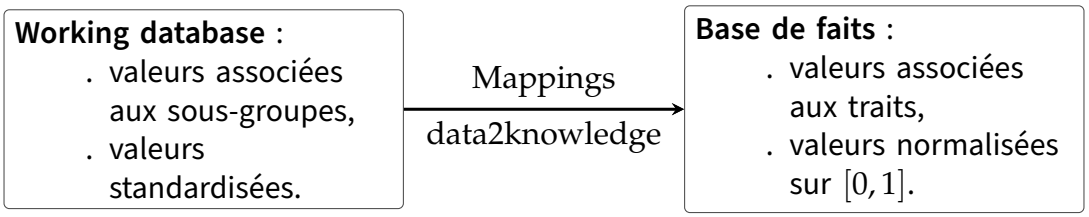


FIGURE 4.9 – Passage de la working database à la base de faits : mappings de data2knowledge

Pour cela, nous avons procédé au préalable à une normalisation des valeurs de traits. Cette étape permet de faire abstraction des différentes unités de mesures afin que les agrégations des valeurs de traits pour calculer une valeur de fonction puis de service aient un sens. La normalisation des traits produit des valeurs décimales dans l'intervalle $[0, \ldots, 1]$. Notre méthode de normalisation des traits se fait en 2 étapes :

- On commence par normaliser les sous-groupes (ou DataNames). En effet, nous avons constaté qu'il est plus pertinent de faire une première étape de normalisation à ce niveau là car la variation des valeurs y est plus faible au niveau des traits. En plus des dataNames, d'autres paramètres sont pris en considération dans la normalisation : l'espèce (AccSpeciesID), les conditions de croissance (GrowingConditions) et la base de données. Nous avons utilisé la formule de normalisation suivante :

$$(V_{moy} - V_{min}) / (V_{max} - V_{min})$$

où V_{moy} est la valeur moyenne du dataName pour l'espèce, dans une même condition de croissance, suivant une même base de données ; V_{min} (respectivement V_{max}) est la valeur minimale (respectivement maximale) du dataName, pour toutes les espèces dans une même condition de croissance

suivant une même base de données.
À la fin de cette étape, pour une unique valeur normalisée pour chaque 4-uplet (ID du sous-groupe, ID de l'espèce, Condition de croissance, Base de données source).

- Une fois la normalisation des dataNames faite, nous passons au niveau des traits. Pour cela, nous prenons une moyenne des valeurs normalisées des dataNames reliés au trait. Nous obtenons donc une valeur normalisée (comprise entre 0 et 1) du trait basée sur une moyenne des valeurs normalisées de ses sous-groupes.
  À ce niveau, pour un même 4-uplet (ID du trait, ID de l'espèce, Condition de croissance, Base de données source) nous avons une valeur normalisée comprise entre 0 et 1.

Une fois que cette normalisation est faite, nous construisons des faits, à l'aide de mappings, dont la forme générale est

$$q_{wDB} \rightarrow q_{FB}$$

où q_{wDB} est une requête de sélection sur la working database (wDB) et q_{FB} insère les faits obtenus à partir des tuples retournés par q_{wDB} dans la base de faits.

Les faits sont construits en utilisant deux prédicats : *aPourValeurTraitInitiale* pour décrire les valeurs de trait, et *aPourEspeceID* pour donner le lien entre un nom d'espèce et son ID dans une base de données source. Ils ont plus précisément la forme suivante :

```
aPourValeurTraitInitiale(ID de trait, ID d'espèce, Condition
de croissance, Valeur normalisée, BD source).
aPourEspeceID(Espece, ID d'espèce, BD source).
```

Ici, Espèce est le nom standard de l'espèce (selon The Plant List) et Espèce ID est son identifiant dans la base de données d'origine (TRY ou DB2). Les faits avec le prédicat `aPourValeurInitiale` considèrent les identifiants d'espèce et de traits dans la base de données d'origine. Comme nous le verrons dans la section suivante, les experts s'expriment avec leurs propres noms de traits, qui seront associés aux identifiants de traits dans les bases de données. Ce sera le rôle des règles d'associer des valeurs aux traits des experts (pour une espèce et une condition de croissance données).

4.2 Acquisition des connaissances expertes

Notre démarche générale consiste à partir de valeurs de traits fonctionnels et à calculer des valeurs de services écosystémiques [MBA22]. Alors que les informations concernant les valeurs des traits fonctionnels pour des espèces de plantes sont présentes dans des sources de données, nous n'avons pas trouvé de sources qui établissent des liens bien définis entre traits, fonctions et services. C'est pourquoi une recherche bibliographique (appuyée par des opinions d'experts) a été conduite pour identifier les liens entre traits et fonctions, et entre fonctions et service.

Schématiquement, notre méthode d'acquisition des connaissances expertes est composée de quatre étapes :

1. définir les services écosystémiques pertinents pour le cas d'étude (voir la section 4.2.1) ;
2. lier ces services aux traits fonctionnels par l'intermédiaire de fonctions (construction de diagrammes), ce qui implique de définir les traits et fonctions pertinents (voir la section 4.2.2) et de choisir une technique d'agrégation ;
3. identifier les bases de données pertinentes et associer les traits du diagramme aux identifiants pertinents dans ces bases de données ; indiquer l'ordre de priorité entre ces bases, globalement ou trait par trait (voir la section 4.2.3) ;
4. formaliser ces connaissances en les traduisant en des faits de la base de connaissances (voir la section 4.2.4).

Notons que des allers-retours entre les étapes 2 et 3 sont nécessaires, afin de trouver des correspondants dans les bases de données des traits présents dans les diagrammes.

4.2.1 Services écosystémiques

La première étape a consisté à définir et choisir les services écosystémiques à prendre en compte. Pour cela, une catégorisation des services a été faite en s'appuyant sur la classification de l'*Efese*[6] (pour *Évaluation Française des Écosystèmes et des Services Écosystémiques*) [Puy].

Trois catégories composées de plusieurs services ont été proposées :

6. www.ecologie.gouv.fr/levaluation-francaise-des-ecosystemes-et-des-services-ecosystemiques

- **Régulation des conditions biophysiques de la production agricole** :
  - Structuration du sol.
  - Fourniture d'éléments nutritifs aux cultures.
  - Stockage et restitution de l'eau aux cultures.
  - Stabilisation des sols et contrôle de l'érosion.

- **Régulation de la qualité biophysique du cadre de vie** :
  - Stockage et restitution de l'eau bleue.
  - Régulation de la qualité de l'eau restituée.
  - Régulation du climat global par atténuation des gaz à effet de serre et stockage de carbone.

- **Régulation biologiques** :
  - Régulation des adventices.
  - Régulation des insectes ravageurs.
  - Régulation des agents pathogènes.
  - Pollinisation des espèces cultivées.

Par la suite, nous avons décidé de commencer notre étude avec les trois premiers services de la première catégorie (régulation des conditions biophysiques de la production agricole) : *structuration du sol* (en anglais *soil structuration*), *fourniture d'éléments nutritifs aux cultures* et *stockage et restitution de l'eau aux cultures*. De plus, les 2 derniers services : "fourniture d'éléments nutritifs aux cultures" et "stockage et restitution de l'eau aux cultures" seront considérés dans un contexte viticole. On parlera donc des services de *fourniture d'azote à la vigne* (en anglais *nitrogen supply to vine*) et de *stockage et restitution de l'eau à la vigne* (en anglais *storage and return of water to the vine*) respectivement.

Ces services à étudier en priorité ont été sélectionnés pour deux raisons : d'une part pour leur pertinence par rapport à la problématique d'enherbement des vignes ; et d'autre part parce qu'on pouvait trouver suffisamment d'éléments dans la littérature en vue d'établir des liens entre ces services et des traits fonctionnels.

4.2.2 Liens trait - fonction - service

Une fois les services d'intérêt sélectionnés, des liens connectant des traits fonctionnels aux fonctions puis au services ont été établis en se basant sur la littérature scientifique en agronomie et en écologie, la littérature grise ainsi que des discussions entre des experts. La synthèse de ces connaissances a pris la forme

de la construction de diagrammes définis sur trois niveaux : traits, fonctions et services (en raccourci : diagrammes TFS). Notons que l'intégration dans le système de tout autre service écosystémique devra passer par la construction de tels diagrammes reliant les traits aux fonctions puis au service souhaité.

Les figures 4.10, 4.11 et 4.12 montrent les diagrammes construits, reliant les services de fourniture d'azote à la vigne, de stockage et restitution de l'eau aux cultures et de structuration du sol aux traits fonctionnels en passant par des fonctions. Notons que les noms des services, fonctions et traits fonctionnels sont en anglais dans nos figures, puisque les bases de données mobilisées contiennent des noms de traits uniquement en anglais.

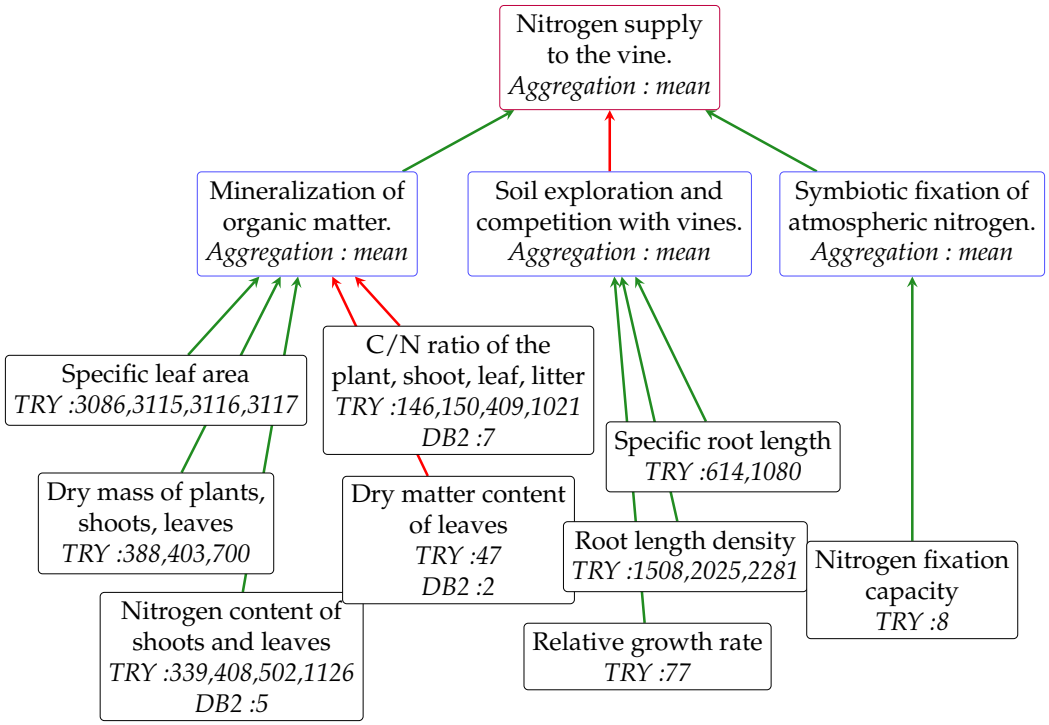


FIGURE 4.10 – Diagramme Trait-Fonction-Service pour le service "Nitrogen supply to the vine". Les traits, fonctions et service sont encadrés en noir, bleu et violet respectivement. Les arêtes en verts et en rouges désignent respectivement des liens positifs et négatifs. Les nombres sont les IDs correspondants aux traits dans les bases de données TRY and DB2.

Dans ces trois figures (4.10, 4.11 et 4.12), les rectangles en noir représentent les traits fonctionnels, ceux en bleu les fonctions et celui en rose le service considéré. Nous notons également qu'il y a deux types de liens :

- Les liens **positifs** (en vert) qui veulent dire : plus la valeur du trait est élevée, mieux la fonction est rendue (de même, des fonctions aux services : plus la valeur de la fonction est élevée, mieux le service est rendu).

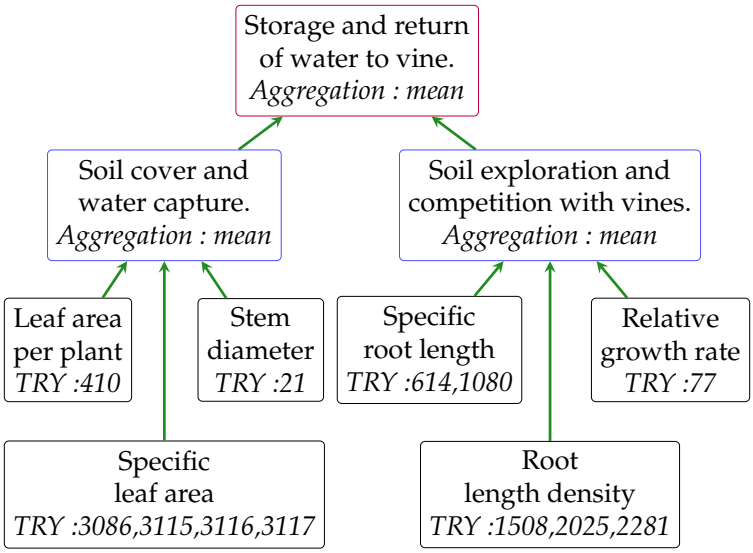


FIGURE 4.11 – Diagramme Trait-Fonction-Service pour le service "Storage and return of water to vine".

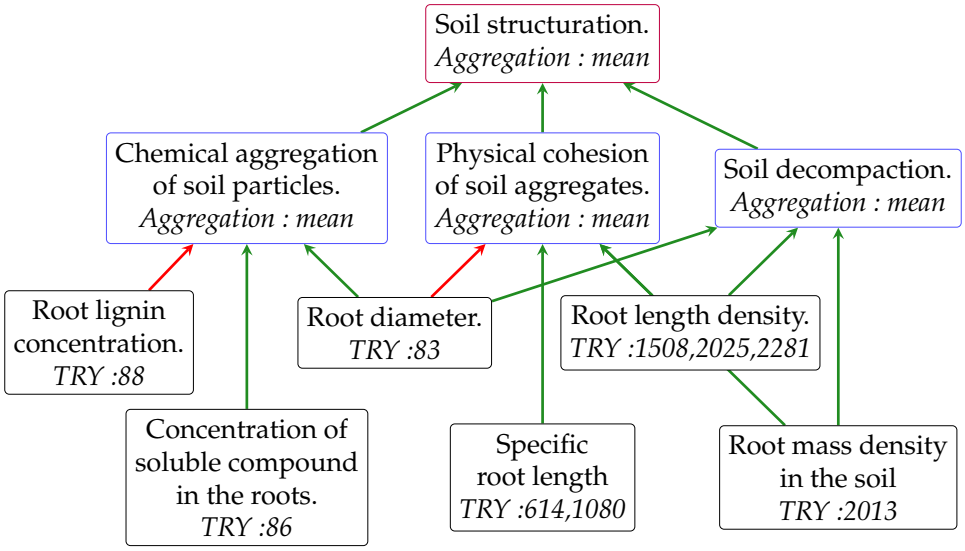


FIGURE 4.12 – Diagramme Trait-Fonction-Service pour le service "Soil structuration".

- Les liens **négatifs** (en rouge) qui veulent dire : plus la valeur du trait est faible, mieux la fonction est rendue (de même, des fonctions aux services : plus la valeur de la fonction est faible, mieux le service est rendu).

Les liens des diagrammes Traits-Fonctions-Services sont basés sur la littérature. Par exemple, si on considère le service “Nitrogen supply to the vine”, ce dernier peut être rendu par des plantes au travers de deux fonctions principalement : *mineralization of organic matter* et *symbiotic fixation of atmospheric nitrogen* et une troisième fonction qui joue un rôle supplémentaire dans la disponibilité

de l'azote, à savoir, *soil exploration and competition with vines* permettant de déterminer le niveau de compétition pour les ressources du sol entre les cultures de service et la vigne.

Le niveau d'activation de ces fonctions est corrélé aux valeurs de certains traits fonctionnels des espèces végétales. Une libération élevée d'azote par minéralisation est associée à des valeurs élevées de surface foliaire spécifique (trait *specific leaf area*), de biomasse (trait *dry mass of plants, shoots, leaves*) et de teneur en azote des plantes (trait *nitrogen content of shoots and leaves*) ; et à des valeurs faibles de teneur en matière sèche foliaire (trait *dry matter content of leaves*) et de rapport carbone/azote (trait *C/N ratio of the plant, shoot, leaf, litter*) [Aba+19 ; Dam+15 ; Han+20]. Une exploration efficace du sol par les racines et donc une forte compétition avec la culture principale est associée principalement à des valeurs élevées de longueur spécifique des racines (trait *specific root length*) et de densité de longueur des racines (trait *root length density*)[Dam+15 ; FW16].

Les figures des deux autres services ont été construites de manière similaire, en se basant sur la littérature.

De plus, les experts ont spécifié les méthodes d'agrégation pour passer des traits aux fonctions (respectivement des fonctions aux services). La méthode d'agrégation est précisée sur les diagrammes Traits-fonctions-service en dessous du nom du service et des fonctions. Par exemple, pour la figure 4.10 le service est calculé est prenant la moyenne (*Aggregation : mean*) des valeurs de fonctions.

4.2.3 Identification des traits des diagrammes avec des traits dans les bases de données

Sous les noms des traits, sont présentes les correspondances avec les bases de données. Par exemple, dans la figure 4.10, on peut noter que le trait *Nitrogen content of shoots and leaves* a des correspondances dans TRY (qui correspondent aux ID 339, 408, 502, 1126) et une correspondance dans DB2, qui correspond à l'ID 5.

Comme on le constate, un même trait peut avoir plusieurs correspondances dans une même base (notons que c'est le cas pour la plupart des traits). En effet, selon les espèces et les traits, ainsi que les ensembles de données d'origine, des mesures selon différentes techniques peuvent être disponibles et il n'y a pas de technique universellement préférée. Par exemple, le trait "Specific leaf area" est lié à 4 traits dans TRY, qui sont tous jugés pertinents :

- Trait ID 3086 correspondant au trait "Leaf area per leaf dry mass (specific

leaf area, SLA or 1/LMA) petiole, rhachis and midrib excluded ".

- Trait ID 3115 correspondant au trait "Leaf area per leaf dry mass (specific leaf area, SLA or 1/LMA) : petiole excluded".
- Trait ID 3116 correspondant au trait "Leaf area per leaf dry mass (specific leaf area, SLA or 1/LMA) : petiole included".
- Trait ID 3117 correspondant au trait "Leaf area per leaf dry mass (specific leaf area, SLA or 1/LMA) : undefined if petiole is in- or excluded".

Lorsqu'un expert fait correspondre plusieurs IDs à un même trait, dans une même base, ces IDs sont appelés *interchangeables*. Évidemment, selon les techniques de mesure, la valeur du trait mesurée pour une plante spécifique n'est pas la même, et ces valeurs, issues de techniques de mesures différentes, ne peuvent pas être directement comparées, ni agrégées ; d'où la nécessité de considérer des valeurs normalisées plutôt que les valeurs brutes pour passer des valeurs de la base de données aux faits de la base de connaissances. Plus d'informations sur la méthode de normalisation faite est donnée dans la section 4.1.5.

De plus, l'intérêt des traits interchangeables est de contribuer à pallier le problème de valeurs manquantes déjà évoqué. En effet, en faisant correspondre plusieurs IDs à un même trait, on augmente le nombre de taxons considérés pour ce trait là.

La table 4.1 contient les noms des traits fonctionnels (colonne "Traits fonctionnels") du service "Nitrogen supply to the vine" (voir figure 4.10) qui ont plus d'une correspondance dans TRY. Les IDs correspondants aux traits sont dans la colonne "*IDs dans TRY*", et, pour chaque ID, nous donnons le nombre de taxons qu'il recouvre (colonne "# taxons ID"). La dernière colonne (colonne "# taxons Trait"), nous donne le nombre de taxons (différents) que recouvrent tous les IDs d'un même trait. Considérons par exemple le trait *specific leaf area*. Ce trait peut correspondre à 4 différents traits dans TRY qui ont pour ID : 3086, 3115, 3116 et 3117. Chaque ID couvre respectivement 403, 7485, 6705 et 12584 taxons dans TRY. Si on avait décidé d'identifier ce trait à un seul correspondant dans TRY, on aurait considéré au mieux 12584 taxons (avec le trait ID 3117). Par contre, en autorisant une correspondance multiple (avec normalisation des valeurs) on arrive à un total de 16006 taxons intégrés dans l'étude, soit un gain d'un peu plus de 3400 taxons (ou un gain de 27.2%).

Nous remarquons également que le gain de taxons est significatif dans le cas des autres traits considérés ayant plusieurs correspondances. En effet, au mini-

Traits fonctionnels	IDs dans TRY	# taxons ID	# taxons Trait
Specific leaf area	3086	403	16006
	3115	7485	
	3116	6705	
	3117	12584	
Dry mass of plants, shoots, leaves	388	68	883
	403	750	
	700	227	
Nitrogen content of shoots and leaves	339	196	333
	408	126	
	502	2	
	1126	115	
C/N ratio of the plant, shoot, leaf, litter	146	4717	6205
	150	155	
	409	2050	
	1021	16	
Root length density	1508	24	80
	2025	51	
	2281	27	
Specific root length	614	1062	1369
	1080	494	

TABLE 4.1 – Nombre d'espèces par trait

mum, nous avons un gain d'environ 17.7% de taxons avec le trait "Dry mass of plants, shoots, leaves" et au maximum un gain d'environ 70% de taxons avec le trait "Nitrogen content of shoots and leaves".

Enfin, notre modélisation inclut des préférences entre bases de données (la forme des faits pour la préférence est donnée en figure 4.14) : il existe un ordre total global par défaut, qui peut être écrasé pour des préférences sur des traits spécifiques (ici, TRY est globalement préféré à DB2). Cela permet de donner une priorité plus élevée à une source plus fiable. Ensuite, la valeur d'un trait pour une espèce est donnée par la source de données la plus prioritaire qui fournit cette valeur. La priorité entre les bases est gérée par les règles, voir section 5.1.4

4.2.4 Construction des faits à partir des diagrammes TFS

La construction des faits logiques à partir des diagrammes TFS s'est faite en passant par les *graphes conceptuels*. Les graphes conceptuels constituent un langage de représentation des connaissances, dans lequel les connaissances sont encodées sous forme de graphes étiquetés (au sens de la théorie des graphes) et dotés d'une traduction en logique du premier ordre [Sow84; CM09]. Nous les

avons utilisés comme une représentation intermédiaire entre les diagrammes TFS et la logique, qui a l'avantage d'être à la fois facile à comprendre par les concepteurs des diagrammes et formelle. Pour dessiner et gérer les graphes conceptuels, nous avons utilisé l'outil visuel *Cogui*[7], qui implémente le cadre développé dans [CM09]. Ce cadre permet de définir des bases de connaissances composées d'un vocabulaire ontologique, de faits et de règles construits sur ce vocabulaire, et de faire des raisonnements sur ces connaissances. Cependant, il ne comporte pas de négation ni de fonctions calculées, ce qui est nécessaire dans certaines de nos règles. Nous avons donc utilisé Cogui uniquement pour construire les faits experts, puis les avons exportés dans le cadre logique (en utilisant la fonctionnalité d'exportation de Cogui).

Le vocabulaire ontologique se compose de quelques concepts organisés en hiérarchie, voir figure 4.14. Cette figure montre les différents concepts utilisés dans notre étude notamment : *objetTFS* (avec les sous-concepts *trait*, *fonction* et *service*), agregation, et source, (avec les sous-concepts *sourceDeDonnee* et *sourceUtilisateur*).

Les relations, quant à elles, sont décrites dans la figure 4.13. Pour décrire les diagrammes experts, seules les relations énumérées aux points 1 et 2 de la figure sont utilisées ; les relations énumérées au point 3 sont utilisées dans les requêtes de l'utilisateur final.

Un graphe conceptuel décrit des entités et les relations entre ces entités. Il est composé de deux types de noeuds : des noeuds concepts (représentant des entités) avec une étiquette de la forme $type$: $marqueur$, où $type$ est un concept et $marqueur$ est soit une entité connue (c'est-à-dire une constante) soit le symbole $\star$ dénotant une entité inconnue (ici, nous n'utilisons que des constantes) ; et des noeuds relations étiquetés par une relation. Chaque nœud relation étiqueté par une relation d'arité k est incident à k arêtes, étiquetées de 1 à k, qui le relient aux nœuds concepts.

7. https://www.lirmm.fr/cogui/.

1**. Relations communes aux diagrammes et aux requêtes de l'utilisateur final**

```
estLieAFonctionAvecPonderation(trait,fonction,ponderation)
estLieAFonction(trait,fonction)
  % spécialisations
    % équivalent à
estLieAFonctionAvecPonderation(trait,fonction,1):
    estPositivementLieAFonction(trait,fonction)
    % equivalent to
estLieAFonctionAvecPonderation(trait,fonction,-1):
    estNegativementLieAFonction(trait,fonction)
estLieAServiceAvecPonderation(fonction,service,ponderation)
estLieAService(fonction,service)
  % spécialisations
    % équivalent à
estLieAServiceAvecPonderation(fonction,service,1):
    estPositivementLieAService(fonction,service)
    % équivalent à
estLieAServiceAvecPonderation(fonction,service,-1):
    estNegativementLieAService(fonction,service)
aPourMethodeAgregation(objetTFS,agregation)
```

2**. Relations dédiées à la construction des diagrammes TFS**

```
aPourTraitID(trait, iD, sourceDeDonnee)
basePrioritaire(sourceDeDonnee, sourceDeDonnee)   % la première base
est considérée comme globalement prioritaire sur le seconde
traitPrioritaireBase(trait, sourceDeDonnee, sourceDeDonnee)  % pour
un trait en particulier la priorité peut changer
```

3**. Relations dédiées aux requêtes de l'utilisateur final**

```
aPourValeurTraitBase(trait,espece,growCond,valeur,source)
aPourValeurTrait(trait,espece,growCond,valeur)
aPourValeurFonction(fonction,espece,valeur,fiabilite,growCond)
aPourValeurFonctionUser(fonction,espece,valeur,fiabilite,growCond)
aPourValeurFonctionCalculee(fonction,espece,valeur,fiabilite,growCond)
aPourValeurService(service,espece,valeur,fiabilite,growCond)
aPourValeurServiceCalculee(service,espece,valeur,fiabilite,growCond)
aPourValeurServiceUser(service,espece,valeur,fiabilite,growCond)
```

FIGURE 4.13 – Relations du vocabulaire ontologique. Les noms donnés aux arguments de la relation reflètent les signatures de la relation (c'est-à-dire les concepts qui typent les arguments), sauf pour les valeurs littérales : ponderation, valeur et fiabilite qui sont des nombres décimaux et iD qui est une chaîne de caractères.

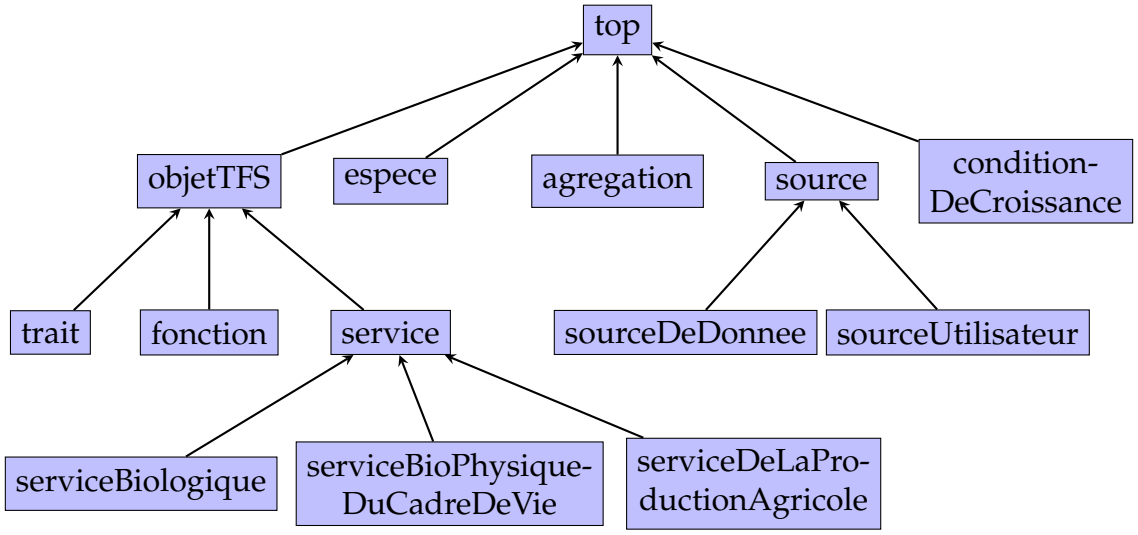
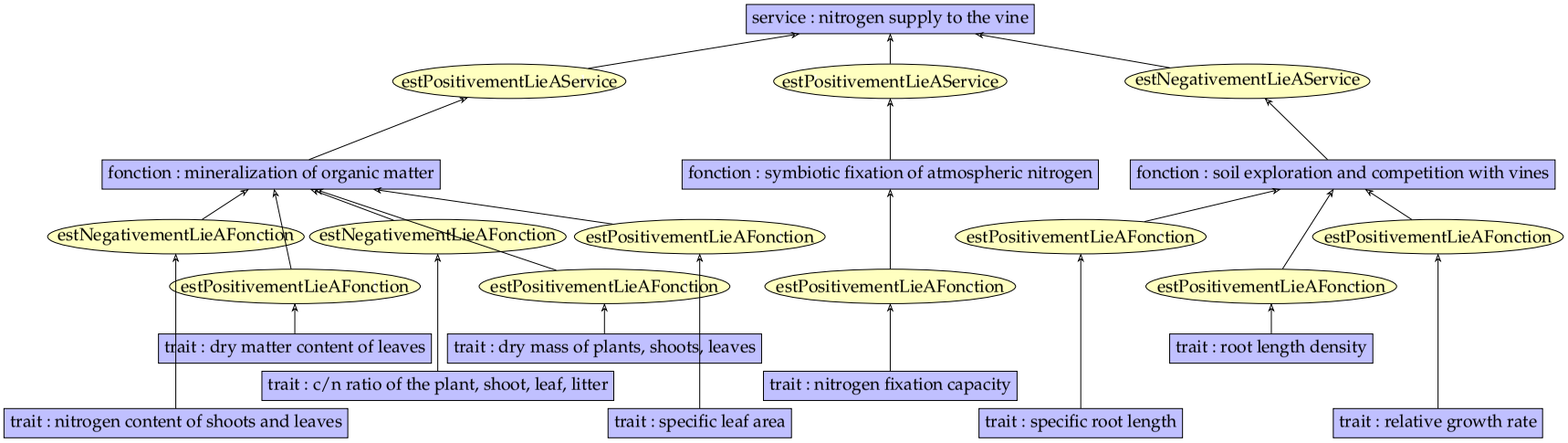
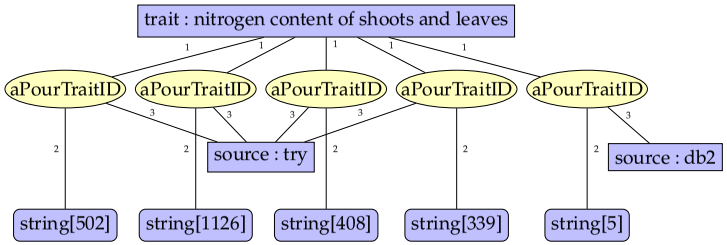


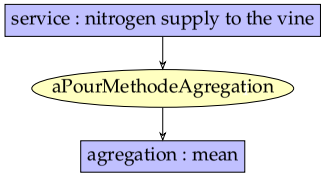
FIGURE 4.14 – Hiérarchie de concepts



(a) Diagramme Trait-Fonction-Service. Les rectangles (violets) représentent des noueds concepts et les ovals (jaunes) des noeuds relations.



(b) Correspondance du trait “Nitrogen content of shoots and leaves” avec les bases de données



(c) Méthode d’agrégation pour calculer une valeur de service

FIGURE 4.15 – Graphe conceptuel associé au diagramme TFS de la figure 4.10. Les rectangles aux côtés arrondis sont associés aux types de données.

La figure 4.15 représente des graphes conceptuels qui traduisent partiellement le diagramme de la figure 4.10. Les noeuds concepts sont représentés par des *rectangles* dans le graphe et les noeuds relations par des *ovales*. Dans le cas de relations binaires (voir le graphe du haut), Cogui remplace les arêtes d'étiquette 1 et 2 par des arcs. Le graphe du haut décrit les relations trait-fonction-service. Les deux graphes du bas traduisent les liens entre le trait "nitrogen content of shoots and leaves" et les ID de traits associés dans TRY et BD2 (graphe b) puis la méthode d'agrégation choisie pour passer des fonctions aux services (graphe c). Des graphes similaires représentent les liens entre traits et fonctions.

Notons que des graphes conceptuels différents sont implicitement joints sur les nœuds qui partagent la même constante. Ceci nous permet, de manière générale, de construire différents graphes qui seront par la suite fusionnés (à partir des mêmes noms de constante) en un même graphe. La possibilité de diviser les graphes en plusieurs composantes facilite l'activité de conception et offre une meilleure lisibilité.

Un graphe conceptuel admet une traduction naturelle en un ensemble de faits logiques. Les concepts et les relations du vocabulaire sont vus comme des prédicats : prédicats unaires pour les concepts et prédicats de la même arité pour les relations. Donc, chaque noeud concept d'étiquette *type :constante* produit un fait de la forme *type(constante)*, et chaque noeud relation étiqueté par une relation r d'arité k produit un fait de la forme $r(c_1, ..., c_k)$, où chaque c_i est la constante correspondant au i-ème voisin du noeud (c'est-à-dire celui à l'extrémité de l'arête de numéro i).

La figure 4.16 liste l'ensemble de faits formant la traduction logique des graphes de la figure 4.15.

```
% Graphe du haut
service("nitrogen supply to the vine").
fonction("mineralization of organic matter").
fonction("soil exploration and competition with vines").
fonction("symbiotic fixation of atmospheric nitrogen").
estPositivementLieAService("mineralization of organic
matter","nitrogen supply to the vine").
estNegativementLieAService("soil exploration and competition with
vines","nitrogen supply to the vine").
estPositivementLieAService("symbiotic fixation of atmospheric
nitrogen","nitrogen supply to the vine").
...
% Graphes du bas
trait("nitrogen content of shoots and leaves").
sourceDeDonnee(try).
sourceDeDonnee(db2).
aPourTraitID("nitrogen content of shoots and leaves","339",try).
aPourTraitID("nitrogen content of shoots and leaves","408",try).
aPourTraitID("nitrogen content of shoots and leaves","502",try).
aPourTraitID("nitrogen content of shoots and leaves","1126",try).
aPourTraitID("nitrogen content of shoots and leaves","5",db2).
aPourMethodeAgregation("nitrogen supply to the vine",fct:mean).
```

FIGURE 4.16 – Traduction (partielle) en logique de la figure 4.15

CHAPITRE 5

Raisonnements et évaluation empirique

Dans le chapitre précédent, nous avons expliqué comment construire la base de faits initiale, composée de faits de données et de faits d'experts. Ce chapitre est consacré aux raisonnements, qui sont effectués par application de règles sur la base de faits initiale jusqu'à saturation de la base de faits, ainsi qu'à une évaluation de la validité applicative des résultats obtenus. Dans la section 5.1, nous décrivons l'ensemble des règles qui composent notre système, en les regroupant par finalité. La section 5.2 présente brièvement l'interrogation de la base de connaissances. Enfin, la section 5.3 détaille l'évaluation empirique des résultats obtenus pour l'un des trois services éco-systémiques considérés.

5.1 Base de règles

Cette section présente les différentes règles de la base de connaissances, en les regroupant selon leur finalité :

1. règles traduisant l'ontologie de domaine ;
2. règles permettant de regrouper les faits issus des diagrammes d'expert en des faits plus complexes reposant sur des prédicats d'arité supérieure ;
3. règles permettant d'associer des valeurs de trait, dites "de base", aux espèces végétales, à partir des identifiants de traits interchangeables ; chaque valeur de trait est alors relative à une source de données ;

```
(R1): sourceDeDonnee(S) → source(S)
(R2): sourceUser(S) → source(S)
```

FIGURE 5.1 – Spécialisation entre concepts. Ces règles déclarent que les concepts *sourceDeDonnee* et *sourceUser* sont des spécialisations du concept *source*

```
(R1): estPositivementLieAFonction(Trait,Fonction) →
estLieAFonction(Trait,Fonction)
(R2): estNegativementLieAFonction(Trait,Fonction) →
estLieAFonction(Trait,Fonction)
(R3): estLieAFonctionAvecPonderation(Trait,Fonction,1) →
estPositivementLieAFonction(Trait,Fonction)
(R4): estPositivementLieAFonction(Trait,Fonction) →
estLieAFonctionAvecPonderation(Trait,Fonction,1)
(R5): estLieAFonctionAvecPonderation(Trait,Fonction,-1) →
estNegativementLieAFonction(Trait,Fonction)
(R6): estNegativementLieAFonction(Trait,Fonction) →
estLieAFonctionAvecPonderation(Trait,Fonction,-1)
```

FIGURE 5.2 – Spécialisation et équivalence entre relations. Les règles $(R1)$ et $(R2)$ déclarent que les relations *estPositivementLieAFonction* et *estNegativementLieAFonction* sont des spécialisations de la relation *estLieAFonction*. Les règles $(R3)$, $(R4)$ et $(R5)$, $(R6)$ expriment l'équivalence.

4. règles permettant d'associer des valeurs de traits, dites "finales", aux espèces, en exploitant les préférences entre sources de données ;
5. règles permettant de calculer les valeurs de fonction et de service des espèces, par agrégation de leurs valeurs de traits (finales).

5.1.1 Règles de l'ontologie

L'ontologie de domaine fournit le vocabulaire des utilisateurs experts (formalisation des diagrammes) et des utilisateurs finaux (formulation des requêtes). L'ensemble de ses concepts et relations a été donné au chapitre précédent, voir les figures 4.13 et 4.14. Les relations sémantiques entre ces éléments sont décrites par des règles simples, exprimant plus précisément la spécialisation entre concepts (figure 5.1), la spécialisation ou l'équivalence entre relations (figure 5.2) ainsi que les signatures des relations (voir figure 5.3).

L'ensemble des règles décrivant l'ontologie est donné en annexe B.

`(R1): estLieAFonction(Trait,Fonction)` $\rightarrow$ `trait(Trait)`

`(R2): estLieAFonction(Trait,Fonction)` $\rightarrow$ `fonction(Fonction)`

FIGURE 5.3 – Signature de relation. Ces règles précisent que la relation *estLieAFonction* prend en premier paramètre un *trait* et en second une *fonction*

5.1.2 Règles de regroupement des faits experts

Pour raisonner sur les faits, en particulier pour calculer diverses agrégations de valeurs, nous avons besoin de connaître l'ensemble des traits (resp. fonctions) liés à une fonction (resp. service) et la cardinalité de cet ensemble. Il en va de même pour l'ensemble des identifiants de traits associés à un trait d'expert. C'est pourquoi certaines règles opèrent sur des faits experts pour produire des faits de la forme suivante :

- `estLieAkTraits(Fonction,Trait1,Poids1,...,Traitk,Poidsk, Agregation)` qui lie une fonction écosystémique (`Fonction`) à ses k traits sous-jacents (`Trait1 ...Traitk`), avec (`Poids1 ...Poidsk`) les poids associés et `Agregation` la méthode d'agrégation de ces valeurs de traits. Ici, le prédicat est d'arité $2k + 2$. Les traits sont ordonnés par ordre lexicographique décroissant.
- `estLieAkFonctions(Service,Fonction1,Poids1,...,Fonctionk,Poidsk, Agregation)` qui est construit de manière similaire pour lier un service à ses k fonctions sous-jacentes. Les fonctions sont ordonnées par ordre lexicographique décroissant.
- `estLieAkTraitIDs(Trait, TraitID1, ..., TraitIDk, SourceDeDonnee, Agregation)` pour lier un trait à ses k IDs de traits interchangeables dans une source de données. Les identifiants de traits sont ordonnés par ordre lexicographique décroissant. Ici, `SourceDeDonnee` désigne la source de données considérée et `Agregation` la méthode d'agrégation des valeurs associées aux ID de traits.

Notons que ces relations de haute arité auraient pu être directement encodées dans les graphes conceptuels associés aux diagrammes TFS, cependant nous avons préféré garder les graphes conceptuels simples et aussi proches que possible des diagrammes TFS construits au préalable. Le passage des relations utilisées par les experts aux relations de haute arité est géré par un ensemble de règles. La figure 5.4 nous donne une partie de ces règles qui permettent de connecter les deux vocabulaires, elles permettent ici de construire les faits de la forme *estLieAkTraits(Fonction,Trait1,Poids1,...,Traitk,Poidsk,Agregation)*, pour k allant de 1

à 3 (ces règles s'étendent naturellement pour les cas où $k > 3$, voir annexe B).

Pour passer à ces relations de haute arité, nous avons introduit des prédicats intermédiaires de la forme *traitsRestantsk* et *traitMaxk*. L'idée principale est de sélectionner à chaque étape le plus grand trait (trait max) parmi tous les traits restants reliés à une même fonction et de le regrouper avec les traits déjà sélectionnés.

Si une fonction f est liée à n traits, à la k-ième étape, on produit un fait de prédicat *traitMaxk* (pour regrouper les k plus grands traits) et $n - k$ faits de prédicat *traitsRestantsk* (pour chaque trait restant à considérer après l'étape k). Et on produit le fait de prédicat *estLieAnTraits* pour f lorsqu'on a un fait de prédicat *traitMaxn* pour f et qu'on ne peut pas produire de fait de prédicat *traitMax(n+1)* pour f.

Pour illustrer, prenons l'exemple d'une fonction f liée à trois traits t1, t2 et t3 avec les poids respectifs p1, p2 et p3 et tels que t1 < t2 < t3. Autrement dit, nous avons les faits suivants :
`estLieAFonctionAvecPonderation(f,t1,p1)`,
`estLieAFonctionAvecPonderation(f,t2,p2)`,
et `estLieAFonctionAvecPonderation(f,t3,p3)`
ainsi que le fait `aPourMethodeAgregation(f,mean)` si la méthode d'agrégation choisie est la moyenne.

Au premier passage :

- $t3$ étant le plus grand trait, $t1$ et $t2$ sont les traits restants (du niveau 1) : (R1) produit `traitsRestants1(f,t1,p1)` et `traitsRestants1(f,t2,p2)` ;
- ensuite (R2) produit le fait `traitMax1(f,t3,p3)` ;

Au deuxième passage, on récupère $t2$, le deuxième plus grand trait :

- $t1$ est le trait restant (du niveau 2) : (R3) produit donc `traitsRestants2(f,t1,p1)` ;
- on regroupe $t2$ avec $t3$ en appliquant la règle (R4) qui produit le fait `traitMax2(f,t3,p3,t2,p2)` ;

Il ne reste donc plus qu'un seul trait (t1) à considérer qui sera récupéré au troisième passage pour construire le fait : `traitMax3(f,t3,p3,t2,p2,t1,p1)` par la règle (R6). Dès lors nous avons regroupé 3 traits reliés à la fonction f, et nous n'en avons pas d'autre, ainsi la règle (R9) s'applique et permet de construire le fait final `estLieA3traits(f,t3,p3,t2,p2,t1,p1,mean)`.

Remarquons que cette façon de procéder, qui exploite un ordre total sur les traits, permet d'avoir un seul fait décrivant l'association entre une fonction et l'ensemble de ses traits, ce qui évite par la suite de faire plusieurs fois les mêmes

calculs.

D'autres règles similaires sont mises en place pour la construction des faits de la forme `estLieA`*k*`Fonctions(Service,Fonction1,Poids1,...,Fonction`*k*`,Poids`*k*`, Agregation)` et `estLieA`*k*`TraitIDs(Trait, TraitID1, ..., TraitID`*k*`, SourceDeDonnee, Agregation)`. Voir annexe B.

5.1.3 Calcul des valeurs de traits en regroupant les IDs interchangeables

La base de faits contient des valeurs de traits normalisées pour les espèces végétales respectant les critères de sélection du cas d'étude (dans notre cas d'étude, le seul critère de sélection est la limitation aux espèces herbacées), et ce, suivant une condition de croissance (`growing condition`). Elle contient également les connaissances des experts liant ces traits aux fonctions puis aux services. Cependant, les valeurs normalisées initiales associées aux IDs des traits de différentes sources de données ne sont pas directement utilisées pour le calcul des valeurs de fonctions. En effet, certaines relations entre les traits sont susceptibles de consolider ces valeurs, notamment en compensant les valeurs manquantes. Comme nous l'avons vu dans la section 4.2.2 du chapitre précédent, chaque trait du diagramme peut avoir un ou plusieurs IDs de correspondance dans la même base de données, ces IDs étant dits interchangeables. Nous rappelons l'importance de la multiplicité de ces ID interchangeables vu le nombre de valeurs manquantes ; en effet, en considérant plusieurs correspondances d'un trait dans une même base de données, on augmente le nombre d'espèces retrouvées (voir la figure 4.1).

A titre d'exemple, nous donnons ci-dessous (figure 5.5) les règles traitant le cas où le trait d'un expert a deux correspondances dans une base de données. Pour un trait *Trait* donné qui correspond à deux IDs de trait de la base de données (*TraitID1* et *TraitID2*), une espèce *SpeciesID*, une condition de croissance *GrowingCondition*, tous issus de la même base de données *DB*, trois cas sont considérés : (i) soit les deux traits de la base de données ont des valeurs et la valeur calculée est l'agrégation de ces valeurs : c'est la règle (R1) de la figure 5.5 qui s'applique ; (ii) soit seulement le premier ID du trait (`TraitID1`) a une valeur dans la base de données, c'est alors sa valeur qui est retenue comme valeur du trait, règle (`R2`), de la même figure ; (iii) soit seulement le second ID du trait (`TraitID2`) a une valeur dans la base de données et c'est donc sa valeur qui est retenue comme valeur du trait, règle (R3), de la même figure.

Notons que dans ces règles, `Agregation` est une variable qui va recevoir un nom de méthode d'agrégation lors de l'application de la règle — par exemple la

```
(R1):estLieAFonctionAvecPonderation(T1, F, P1),
estLieAFonctionAvecPonderation(T2, F, P2), T1 < T2
→ traitsRestants1(F, T1, P1).

(R2):estLieAFonctionAvecPonderation(T1, F, P1), not
traitsRestants1(F, T1, P1)
→ traitMax1(F, T1, P1). % T1 est le trait max pour F

(R3):estLieAFonctionAvecPonderation(T1,F,P1),
estLieAFonctionAvecPonderation(T2,F,P2), T1 < T2, traitMax1(F,T3,P3),
T3 ≠ T2
→traitsRestants2(F, T1, P1).

(R4):traitMax1(F,T1,P1), estLieAFonctionAvecPonderation(T2,F,P2), T2
< T1, not traitsRestants2(F,T2,P2)
→traitMax2(F, T1, P1, T2, P2).

(R5):estLieAFonctionAvecPonderation(T1,F,P1),
estLieAFonctionAvecPonderation(T2,F,P2), T1 < T2,
traitMax2(F,T3,P3,T4,P4), T3 ≠ T2, T4 ≠ T2
→ traitsRestants3(F, T1, P1).

(R6):traitMax2(F,T1,P1,T2,P2),
estLieAFonctionAvecPonderation(T3,F,P3), T1≠T3, T2 ≠ T3,
not traitsRestants3(F,T3,P3)
→ traitMax3(F, T1, P1, T2, P2, T3, P3).

(R7):traitMax1(F, T1, P1), not traitMax2(F, T1, P1, T2, P2)
→ estLieA1Trait(F, T1, P1).

(R8)traitMax2(F, T1, P1, T2, P2), not traitMax3(F, T1, P1, T2, P2,
T3, P3), aPourMethodeAgregation(F,Ag)
→ estLieA2Traits(F, T1, P1, T2, P2, Ag).

(R9):traitMax3(F, T1, P1, T2, P2, T3, P3), not traitMax4(F, T1, P1,
T2, P2, T3, P3, T4, P4), aPourMethodeAgregation(F,Ag)
→ estLieA3Traits(F, T1, P1, T2, P2, T3, P3, Ag).
```

FIGURE 5.4 – Quelques règles permettant de passer des relations "simples" utilisées dans la formalisation des diagrammes TFS aux relations de haute arité utilisées en interne pour faciliter le raisonnement. Ces règles permettent ici la construction des faits de la forme *estLieAkTraits(Fonction,Trait1,Poids1,...,Traitk,Poidsk,Agregation)*, pour k allant de 1 à 3. F = Fonction, T = Trait et P = Poids.

```
(R1): estLieA2TraitIDs(Trait, TraitID1,TraitID2, DB, Agregation),
aPourEspeceID(Espece,EspeceID,DB),
aPourValeurTraitInitiale(TraitID1,EspeceID,GrowCond,V1,DB),
aPourValeurTraitInitiale(TraitID2,EspeceID,GrowCond,V2,DB)
→ aPourValeurTraitBase(Trait,Espece,GrowCond,
fct:agreg2(Agregation,V1,V2), DB).

(R2): estLieA2TraitIDs(Trait, TraitID1,TraitID2, DB, Agregation),
aPourEspeceID(Espece,EspeceID,DB),
aPourValeurTraitInitiale(TraitID1,EspeceID,GrowCond,V1,DB),
not aPourValeurTraitInitiale(TraitID2,EspeceID,GrowCond,V2,DB)
→ aPourValeurTraitBase(Trait,Espece,GrowCond,V1,DB).

(R3): estLieA2TraitIDs(Trait, TraitID1,TraitID2, DB, Agregation),
aPourEspeceID(Espece,EspeceID,DB),
not aPourValeurTraitInitiale(TraitID1,EspeceID,GrowCond,V1,DB),
aPourValeurTraitInitiale(TraitID2,EspeceID,GrowCond,V2,DB)
→ aPourValeurTraitBase(Trait,Espece,GrowCond,V2,DB).
```

FIGURE 5.5 – Règles pour agréger les valeurs de traits initiales en une seule valeur de trait dans une même base

constante `mean` désignant la moyenne.

Dans les règles (R2) et (R3), la méthode d'agrégation n'est finalement pas utilisée, puisqu'un seul des deux traits a une valeur ; lors d'une application de la règle (R1), le nouveau fait construit est de la forme `aPourValeurTraitBase(t,e,g,m,d)` où m est le résultat de l'évaluation de la fonction calculée `fct:agreg2` (qui est par exemple appelée avec les paramètres `mean`, v_1 et v_2, où v_1 et v_2 sont les valeurs prises par les variables $V1$ et $V2$).

À la fin de cette étape, nous avons donc une unique valeur de trait pour une espèce dans une même base et suivant une même condition de croissance (growing condition). Ceci est exprimé par des faits de la forme suivante :

$$aPourValeurTraitBase(Trait, Espece, GrowingCondition, ValeurNormalisee, DB).$$

5.1.4 Calcul des valeurs finales des traits

L'objectif de cette étape est d'arriver à une unique valeur de trait pour une espèce, dans une condition de culture donnée (c'est-à-dire une valeur de trait indépendante de la source de données). Pour cela, nous exploitons l'ordre de préférence sur les bases de données pour retenir la première valeur disponible. Ces ordres de préférence ont deux intérêts : premièrement, pallier les valeurs man-

quantes, et deuxièmement, donner une priorité plus élevée à une source plus fiable, par exemple spécialisée dans certains traits spécifiques. En pratique, lorsqu'on a plusieurs bases avec un ordre de préférence, la valeur d'un trait pour une espèce est d'abord recherchée dans la base préférée, si cette valeur est manquante, la recherche se fait dans la deuxième base préférée et ainsi de suite jusqu'à trouver la première base avec une valeur.

Un ordre de préférence entre les bases a été instauré par défaut lors de la construction des diagrammes TFS. Cet ordre choisi entre les bases, est basé sur la pertinence des bases présentes. Dans notre cas nous supposons que TRY est préférée à DB2. Par ailleurs, il est possible par la suite que les concepteurs des diagrammes TFS décident d'instaurer un ordre de préférence différent (de l'ordre par défaut) pour un trait en particulier. L'intérêt est que certaines sources peuvent être plus pertinentes pour des traits sur lesquels elles sont spécialisées.

Dans la suite, nous détaillons ces deux possibilités :

(i) d'une part l'ordre total par défaut qui est instauré entre les bases de données considérées est commun à tous les traits. Supposons que nous ayons 3 bases de données : $db1$, $db2$ et $db3$, et que nous décidions que par défaut, pour tous les traits, $db1$ est préférée à $db2$ (qu'on notera par le suite par $db1 > db2$), elle-même préférée à $db3$ ($db2 > db3$). Ceci est exprimé formellement par les faits : `{basePrioritaire(db1, db2), basePrioritaire(db2, db3)}`

et par la règle de transitivité

$$basePrioritaire(DB1, DB2), basePrioritaire(DB2, DB3) \rightarrow basePrioritaire(DB1, DB3)$$

qui produit le fait `basePrioritaire(db1,db3)`.

(ii) d'autre part, les concepteurs des diagrammes TFS peuvent décider d'un autre ordre total pour un trait donné. Considérons à nouveau les 3 bases $db1$, $db2$ et $db3$ avec le même ordre total par défaut `{basePrioritaire(`$db1, db2$`), basePrioritaire(`$db2, db3$`)}` . Et supposons maintenant, que pour le trait $t1$, nous trouvions que la base $db3$ est plus pertinente, et souhaitions uniquement pour ce trait définir un nouvel ordre de préférence : $db3 > db1 > db2$; ceci se traduit formellement par l'ajout des faits
`{traitPrioritaireBase(`$t1, db3, db1$`), traitPrioritaireBase(`$t1, db1, db2$`)}`
et la règle de transitivité

`traitPrioritaireBase(`$T, DB1, DB2$`), traitPrioritaireBase(`$T, DB2, DB3$`)`
$\rightarrow$ `traitPrioritaireBase(`$T, DB1, DB3$`)`

(R1): aPourValeurTraitBase(Trait,Espece,GrowCond,Valeur1,$DB1$),
aPourValeurTraitBase(Trait,Espece,GrowCond,Valeur2,$DB2$),
basePrioritaire($DB1$,$DB2$), *not* traitPrioritaireBase(Trait,$DB2$,$DB1$)
$\rightarrow$ valeurTraitPrioritaireSur(Trait,Espece,Valeur1,$DB1$,Value2,$DB2$,
GrowCond).

(R2): aPourValeurTraitBase(Trait,Espece,GrowCond,Valeur1,$DB1$),
aPourValeurTraitBase(Trait,Espece,GrowCond,Valeur2,$DB2$),
traitPrioritaireBase(Trait,$DB2$,$DB1$)
$\rightarrow$ valeurTraitPrioritaireSur(Trait,Espece,Valeur2,$DB2$,Valeur1,$DB1$,
GrowCond).

(R3): aPourValeurTraitBase(Trait,Esp,CC,$V1$,$DB1$),
not valeurTraitPrioritaireSur(Trait,Esp,$V2$,$DB2$,$V1$,$DB1$,CC)
$\rightarrow$ aPourValeurTrait(Trait,Esp,$V1$,CC).

FIGURE 5.6 – Règles permettant de gérer l'ordre de préférence entre les bases de données

qui produit le fait traitPrioritaireBase($t1$,$db3$,$db2$).

Cet ordre ne sera pris en compte que pour les valeurs du trait $t1$. Les autres traits suivront l'ordre de préférence par défaut.

Enfin, un ensemble de règles permet de gérer les priorités entre bases de données, voir figure 5.6.

Dans le cas où aucune préférence entre les bases n'a été émise pour un trait en particulier, la règle (R1) s'applique. En effet, on a $basePrioritaire(DB1, DB2)$ qui réflète la priorité par défaut, et aucune priorité contraire pour un trait particulier $not\ traitPrioritaireBase(Trait, DB2, DB1)$. Sinon, dans le cas où un ordre de préférence est donné pour un trait en particulier et est différent de l'ordre par défaut, les autres règles sont appliquées, en commençant par (R2) qui déduit que la valeur $Valeur2$ du trait $Trait$ issue de $DB2$ est prioritaire sur la valeur $Valeur1$ du même trait, issue de $DB1$. Par la suite, (R3) s'applique et permet de garder comme valeur celle présente dans la base préférée. En effet, cette règle s'applique lorsqu'un trait a une valeur pour une espèce dans une certaine base de données (DB) (atome : $aPourValeurTraitBase(Trait, Espece, GrowCond, Valeur, DB)$) et qu'il n'existe pas de base qui soit prioritaire à DB, (atome :
$not\ valeurTraitPrioritaireSur(Trait, Espece, Valeur2, DB2, Valeur, DB, CC)$).

À la fin de cette étape nous produisons des faits donnant des valeurs de traits pour des espèces indépendamment des sources de données. Ces faits sont de la forme

$$aPourValeurTrait(Trait, Espece, Valeur, GrowCond)$$

L'étape de consolidation des valeurs de traits est terminée, nous passons main-

```
% (R1): les deux traits ont des valeurs, 100% fiabilité
estLieA2Traits(Fonction,Trait1,Lien1,Trait2,Lien2,Agregation),
aPourValeurTrait(Trait1,Espece,V1,GrowCond),
aPourValeurTrait(Trait2,Espece,V2,GrowCond)
→ aPourValeurFonctionCalculee(Fonction,Espece,
fct:agreg2Liens(Agregation,V1,Lien1,V2,Lien2),100,GrowCond).

% (R2): seul le premier trait est renseigné, 50% fiabilité
estLieA2Traits(Fonction,Trait1,Lien1,Trait2,Lien2,Agregation),
aPourValeurTrait(Trait1,Espece,V1,GrowCond),
not aPourValeurTrait(Trait2,Espece,V2,GrowCond)
→ aPourValeurFonctionCalculee(Fonction,Espece,V1,50,GrowCond).

% (R3): seul le second trait est renseigné, 50% fiabilité
% règle similaire à (R2)
```

FIGURE 5.7 – Règles permettant de calculer les valeurs de fonction pour une espèce

tenant au calcul des valeurs de fonction puis de service.

5.1.5 Des valeurs de traits finales aux valeurs de fonction puis de service

L'objectif des règles dans cette section est d'exprimer que le score d'une espèce pour une fonction écosystémique est l'agrégation de ses valeurs pour tous les traits participant à cette fonction. Et de la même manière de passer des fonctions écosystémiques aux services. Nous décrirons par conséquent uniquement le passage des traits aux fonctions.

Comme nous l'avons mentionné au chapitre précédemment, nous sommes confrontés ici au problème des valeurs manquantes. En effet, il arrive souvent que toutes les valeurs des traits participant à une fonction ne soient pas renseignées pour certaines espèces. La façon naturelle de procéder a d'abord été d'écarter les espèces pour lesquelles une valeur de trait requise était manquante. Cependant, cela faisait disparaitre presque toutes les espèces lorsqu'on en venait aux services écosystémiques. Nous avons donc décidé de ne pas exclure les espèces pour lesquelles tous les traits n'ont pas de valeur, tout en ajoutant un paramètre de *fiabilité*, correspondant au pourcentage de traits valués parmi l'ensemble des traits attendus (voir aussi la section 4.1.4.2 du chapitre précédent).

A titre d'illustration, la figure 5.7 présente les règles permettant de calculer la valeur d'une fonction écosystémique liée à 2 traits : (R1) considère le cas où les deux traits sont remplis (la fiabilité du résultat est alors de 100%) et (R2-R3) le cas où un seul trait est rempli (alors la fiabilité du résultat est de 50%).

À la fin de cette étape nous produisons des faits de la forme :
`aPourValeurFonctionCalculee(Fonction,Espece,Valeur,Fiabilite,GrowCond)`

Le prédicat *aPourValeurFonctionCalculee* désigne une valeur de fonction qui est calculée à partir des valeurs de traits. En plus de ces valeurs de fonctions qui proviennent des traits, nous pouvons potentiellement avoir des valeurs données par l'utilisateur.

En effet, on suppose que l'utilisateur peut intervenir au niveau des fonctions ; dans ce cas, un fait de la forme `aPourValeurFonctionUser(Fonction,Espece, Valeur,Fiabilite,GrowCond)` sera créé. Nous considérons par la suite que la valeur donnée par l'utilisateur est toujours prioritaire par rapport à la valeur calculée à partir des traits. Notons également que la participation de l'utilisateur se fait en donnant une valeur comprise en 0 (valeur min) et 1 (valeur max) pour une espèce. Si la valeur 1 est donnée pour une espèce, cela signifie que le rendu de la fonction par cette espèce est très élevé, à contrario, une valeur de 0 signifie que le rendu de la fonction par l'espèce est très faible. Par la suite, on pourrait imaginer que l'utilisateur entre une valeur qualitative. En pratique, cette valeur qualitative serait traduite en une valeur quantitative avant d'être utilisée dans les règles. Par exemple, cela permettrait à l'utilisateur de déclarer qu'une espèce donnée rend *moyennement* la fonction, la valeur qualitative "moyen" étant traduite en la valeur 0.5.

Nous considérons également que l'utilisateur peut intervenir au niveau des valeurs de traits. Dans ce cas, l'utilisateur sera considéré comme une source de données qui sera prioritaire par rapport autres bases de données.

Remarquons que l'importance de la contribution de l'utilisateur est double : d'une part, l'utilisateur peut compléter des valeurs manquantes, ce qui permet d'augmenter la fiabilité des résultats ; d'autre part, il peut contredire les valeurs données, par exemple, s'il n'est pas d'accord avec une valeur de fonction pour une espèce calculée à partir des traits, il peut en donner une autre qui sera prise en compte dans le raisonnement.

À partir des formes de faits précédents : *aPourValeurFonctionCalculee(Fonction, Espece, Valeur, Fiabilite, GrowCond)* et *aPourValeurFonctionUser(Fonction, Espece, Valeur, Fiabilite, GrowCond)* et en utilisant les règles de la figure 5.8, nous produisons des faits qui donnent les valeurs (finales) de fonctions pour une espèce donnée et qui sont de la forme

$$aPourValeurFonction(Fonction, Espece, Valeur, Fiabilite, GrowCond)$$

```
(R1): aPourValeurFonctionCalculee(Fonction,Espece,Valeur,Fiabilite,GrowCond),
not  aPourValeurFonctionUser(Fonction,Espece,Valeur,Fiabilite,GrowCond)
→ aPourValeurFonction(Fonction,Espece,Valeur,Fiabilite,GrowCond).

(R2): aPourValeurFonctionUser(Fonction,Espece,Valeur,Fiabilite,GrowCond)
→ aPourValeurFonction(Fonction,Espece,Valeur,Fiabilite,GrowCond).
```

FIGURE 5.8 – Règles permettant de prendre en compte des valeurs de fonctions données par l'utilisateur

Par la suite, ces faits seront utilisés pour calculer des valeurs de services. Et, comme dans le cas des fonctions, nous aurons également 2 prédicats : le prédicat *aPourValeurServiceCalculee* qui donne une valeur calculée à partir d'une agrégation des valeurs de fonctions qui sont reliées au service ; puis le prédicat *aPourValeurServiceUser* qui permet à l'utilisateur de fournir une valeur de service.

Enfin, similairement au calcul des fonctions, les règles permettent de produire des faits de la forme

$$aPourValeurService(Service, Espece, Valeur, Fiabilite, GrowCond)$$

La liste complète des règles est en annexe B.

5.1.6 Stratification de la base de règles

Comme nous l'avons dit en chapitre 3, la saturation d'une base de faits avec un ensemble de règles contenant des règles avec négation se fait strate par strate. Pour déterminer les différentes strates, il faut vérifier que l'ensemble de règles considéré est bien stratifiable. Nous avons vu en section 3.2.3.4 du chapitre 3, qu'une façon simple de vérifier si un ensemble de règles est stratifiable est de représenter son graphe de dépendance des prédicats intensionnels et de vérifier qu'il ne contient aucun circuit avec un arc négatif.

Pour une meilleure compréhension, nous avons décidé de découper ce graphe en plusieurs sous-graphes correspondant aux sous-ensembles de règles 1 à 5 considérés en début de section :

- le sous-graphe $G1$ associé aux règles du groupe 1 (ontologie de domaine) ne comporte que des arcs positifs. Nous n'allons donc pas le représenter ici ;
- le sous-graphe associé aux règles du groupe 2 (permettant de passer des prédicats des diagrammes TFS aux prédicats de haute arité) est composé de plusieurs composantes connexes, qui ont été intégrés dans les graphes $G3$ et $G5$. Il n'y a donc pas de graphe $G2$;

- le sous-graphe $G3$ associé aux règles du groupe 3 (permettant de gérer les traits interchangeables dans une même base) est représenté en figure 5.9 ;
- le sous-graphe $G4$ associé aux règles du groupe 4 (permettant de gérer les priorités entre bases) est représenté en figure 5.10 ;
- le sous-graphe $G5$ associé aux règles du groupe 5 (permettant de calculer les valeurs de fonction et de service) est représenté en figure 5.11 ;

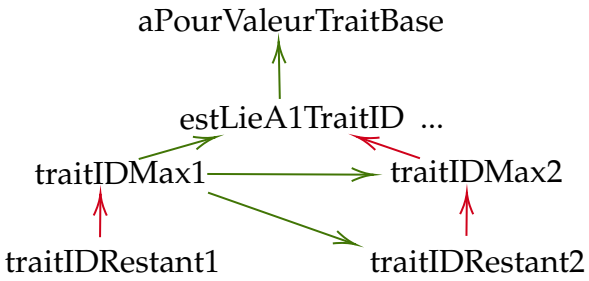


FIGURE 5.9 – Sous-graphe $G3$ correspondant aux règles permettant de gérer la correspondance multiple des traits dans une même base de données.

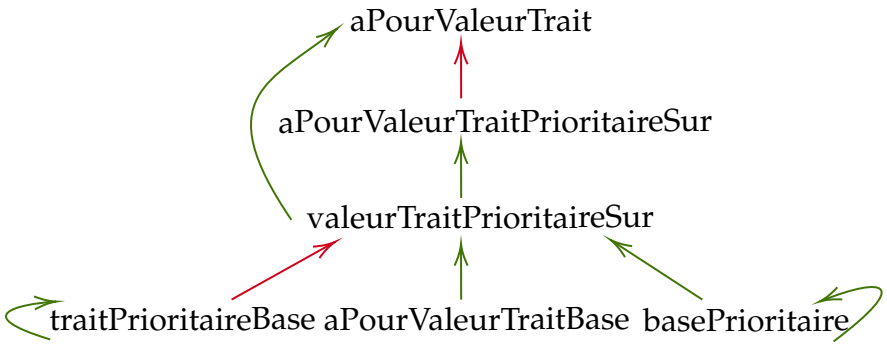


FIGURE 5.10 – Sous-graphe $G4$ correspondant aux règles permettant la gestion des priorités entre bases et le calcul d'une valeur de trait indépendante des bases.

Remarquons qu'aucun sous-graphe ne contient de circuit avec un arc négatif. On peut de plus vérifier que l'union de ces graphes (en fusionnant les sommets communs) n'en contient pas non plus. Nous présentons le graphe global (sans le sous-graphe de l'ontologie) en annexe C.1. Notons que rajouter le sous-graphe de l'ontologie ne rajoutera que des arcs partant des prédicats du sous-graphe global et arrivant à celui de l'ontologie. Il sera donc sans circuit négatif.

5.2 Interrogation du système

Les requêtes de l'utilisateur final sont formulées de manière privilégiée en utilisant le vocabulaire de l'ontologie du domaine, notamment les relations listées aux points 1 et 3 de la figure 4.13. Ce vocabulaire permet d'interroger les

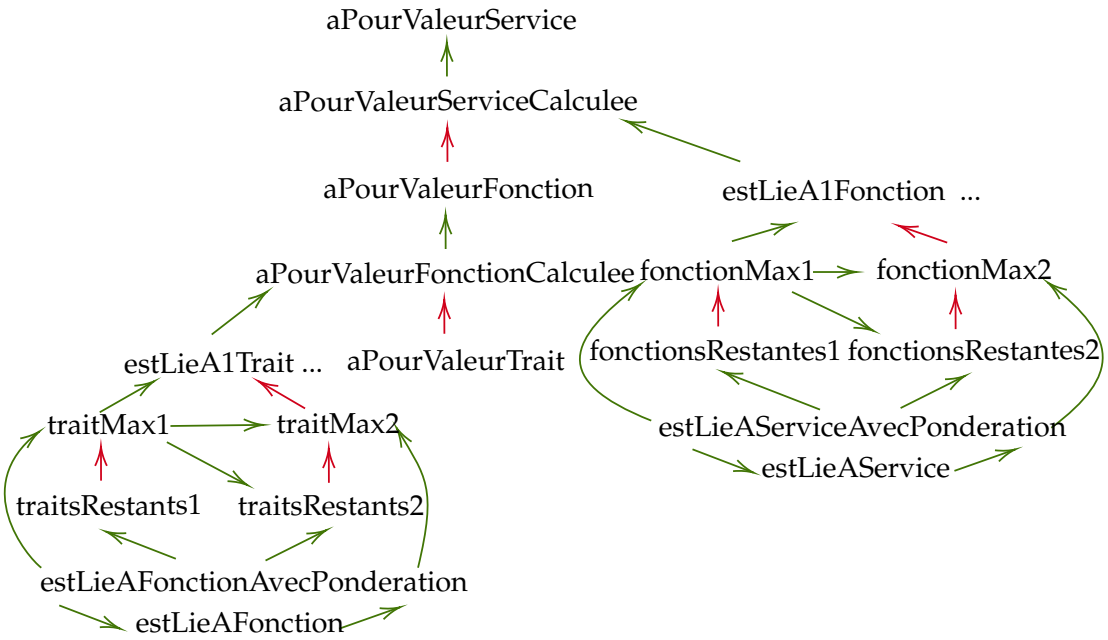


FIGURE 5.11 – Sous-graphe *G*5 correspondant aux règles permettant le calcul des valeurs de fonctions puis de services.

diagrammes TFS (par exemple, quels traits sont négativement liés à une fonction écosystémique elle-même liée au service de "fourniture d'azote à la vigne" ?) ainsi que les contributions des espèces aux fonctions et services de l'écosystème (par exemple, quelles sont les 10 meilleures espèces pour le service "fourniture d'azote à la vigne" avec une fiabilité du résultat d'au moins 60% ?)

Bien que nous distinguions le vocabulaire pertinent pour un utilisateur final, toute la base de faits saturée peut être interrogée. Comme déjà mentionné, la base de faits saturée est stockée dans une base de données relationnelle, ce qui permet de bénéficier de toute la puissance de SQL. Dans le chapitre 3 (section 3.3), nous avons évoqué les "mappings de stockage", que l'on pourrait utiliser pour construire une base de données ne contenant que la portion de la base de faits saturée que l'on désire rendre accessible à un utilisateur.

La base de données comporte 1 relation par prédicat du vocabulaire de la base de faits et les attributs de cette relation correspondent aux arguments de la forme des faits associée à ce prédicat. Par exemple la table *aPourValeurService* sera construite et aura 5 attributs : Fonction, Espece, Valeur, Fiabilite, GrowCond. Pour l'instant, le type des attributs donné par défaut lors de la construction des relations est le type *String*, qui veut dire une chaine de caractères. Notons qu'il est possible d'envisager d'affiner les types en les faisant correspondre aux types des arguments des relations.

Ainsi la requête précédente : *Quelles sont les 10 meilleures espèces pour le ser-*

Espèces	Valeur	Fiabilité
Lotus corniculatus	0.777	89
Trifolium repens	0.765	78
Trifolium pratense	0.757	78
Trifolium hybridum	0.723	55
Trifolium dubium	0.722	67
Lathyrus pratensis	0.719	55
Anthyllis vulneraria	0.718	67
Trifolium campestre	0.705	55
Medicago lupulina	0.7	67
Onobrychis viciifolia	0.695	55

FIGURE 5.12 – Résultat donnant les 10 espèces les mieux classées pour le service de fourniture d'azote à la vigne ainsi que la fiabilité du score.

vice "fourniture d'azote à la vigne" avec une fiabilité du résultat d'au moins 60% *?"* est traduite en la requête SQL suivante :

*SELECT * FROM aPourValeurService WHERE Service = "fourniture azote a la vigne" AND Fiabilité > 50 ORDER BY aPourValeurService.Valeur DESC LIMIT 10.*

La réponse à cette requête [1] est présentée sous forme de table en figure 5.12.

5.3 Évaluation des résultats

De manière générale, l'approche actuelle permet de proposer à l'utilisateur une liste d'espèces de plantes *susceptibles* de rendre un ou plusieurs services écosystémiques voulus et les valeurs de service pour chaque espèce de plante.

Afin d'évaluer la validité applicative de l'approche, nous nous sommes focalisés sur un service écosystémique spécifique pour un ensemble d'espèces sélectionnées a priori, et avons comparé les scores fournis d'une part par l'outil et d'autre part par la littérature du domaine.

Plus précisément, les experts agronomes ont procédé comme suit :

1. *Sélection d'un service* : le service de "fourniture d'azote à la vigne" (figure 4.10) a été choisi pour une première évaluation des résultats. Ce choix se justifie par le fait que c'est l'un des services écosystémiques les plus pertinents pour l'enherbement de la vigne, en outre bien documenté dans la littérature.

1. En l'état actuel de l'implémentation, les réponses sont présentées sous forme de tableaux en utilisant l'interface de gestion de données PhpMyAdmin, https://www.phpmyadmin.net

2. *Sélection des espèces* : sur la base d'une recherche documentaire, un ensemble de 23 espèces herbacées, suffisamment bien documentées dans la littérature scientifique et grise (voir la liste des espèces dans la figure 5.15) a été sélectionné.
3. *Classement des espèces à partir de la littérature* : 16 références agronomiques (colonnes du tableau en figure 5.13 ont participé à classer les espèces, chacune fournissant une comparaison entre certaines des espèces sélectionnées — lignes du tableau — pour le service choisi). Notons que chaque article ne couvre qu'une partie des espèces sélectionnées (entre 2 et 8 espèces). De chaque article, un ordre sur les espèces couvertes a été fait ; formellement, cet ordre est un pré-ordre total, car certaines espèces considérées dans un article peuvent être indiscernables selon cet article et ont alors le même rang.

Étant donné deux espèces s_i et s_j, on note $s_i > s_j$ si s_i est jugé (strictement) meilleure que s_j pour le service selon au moins une référence. Pour obtenir un classement global, nous avons construit un graphe orienté, dont les nœuds sont les espèces s_i et il existe un arc (s_i, s_j) si $s_i > s_j$ dans au moins une référence. Ce graphe est représenté sur la figure 5.14, après suppression de tous les arcs redondants par souci de clarté (un arc (s_i, s_j) est redondant s'il existe déjà un chemin de s_i à s_j dans le graphe).

Nous pouvons constater que les résultats de la littérature sont remarquablement cohérents, puisque le graphe n'a aucun circuit. Par conséquent, le graphe fournit un ordre partiel sur l'ensemble des espèces. Nous avons ensuite considéré toutes les extensions linéaires de cet ordre partiel (c'est-à-dire tous les ordres totaux compatibles avec cet ordre partiel) et avons attribué à chaque espèce un score égal à la moyenne de ses rangs dans les extensions (en commençant par le rang 0).

Par exemple, si on ne considère que le sous-graphe induit par les noeuds 21, 19 et 23, on a $21 > 23$ et $19 > 23$, donc 2 extensions linéaires : $21 > 19 > 23$ et $19 > 21 > 23$. Ainsi, 19 et 21 ont un score de 0.5 (moyenne entre 0 et 1) et 23 le score 2.

Le graphe entier admet plus de 44 millions d'extensions linéaires. Les scores obtenus sont présentés dans le tableau 5.15, colonne "Classement biblio".

D'autre part, la valeur de service calculée par l'outil pour chaque espèce et la fiabilité de ce calcul sont respectivement indiquées dans les colonnes "Valeur par l'outil" et "Fiabilité" du tableau 5.15. Nous rappelons que la fiabilité dépend de la proportion de traits pour lesquels une valeur a pu être récupérée. Les notations

Espèce	(*) itab2012	[RW96]	[Vog+22]	[Kem+22]	[Bra+08]	[KJ02]	[EFB84]	[Mey+22]	[Ben+12]	[KSJ96]	[Dec+94]	[BWR00]	[Cap+04]	[SS91]	[She92]	[KS98]
Avena sativa	–	–	–	–	–	–	–	–	–	–	–	5	–	5	4	–
Brassica napus	–	–	–	–	–	–	–	–	–	4	4	–	–	–	–	–
Crotalaria juncea	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–
Dactylis glomerata	–	–	4	–	–	–	–	–	–	–	–	–	–	–	–	–
Fagopyrum esculentum	–	–	–	7	–	–	–	–	–	–	–	–	–	–	–	–
Festuca arundinaceae	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–
Glycine max	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–
Hordeum vulgare	7	–	–	–	–	–	–	–	2	–	–	–	–	–	–	–
Lolium multiflorum	8	–	–	5	–	3	–	3	–	5	–	–	3	–	–	3
Lotus corniculatus	3	–	–	–	–	–	–	–	–	–	–	1	–	–	–	–
Lupinus albus	–	–	–	3	–	–	–	–	–	–	–	–	–	–	–	–
Medicago lupulina	3	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–
Medicago sativa	6	–	5	–	–	–	–	–	–	–	–	–	–	–	–	–
Melilotus officinalis	–	–	–	–	4	–	–	–	–	–	–	–	–	–	–	–
Plantago lanceolata	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–
Pisum sativum	–	–	–	–	–	–	–	–	–	2	2	3	–	4	2	–
Raphanus sativus	–	–	–	6	–	–	–	–	–	–	–	–	–	–	–	–
Rumex acetosa	–	–	–	2	–	–	–	–	–	–	–	–	–	–	–	–
Secale cereale	–	3	–	–	–	2	2	–	–	3	–	–	–	–	–	2
Sinapsis alba	–	–	–	–	–	–	–	2	–	–	–	4	–	–	–	–
Trifolium alexandrinum	–	–	–	–	–	–	–	–	–	–	–	–	–	2	–	–
Trifolium incarnatum	–	2	3	4	3	–	–	–	–	–	3	–	–	–	–	–
Trifolium pratense	1	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–
Trifolium repens	3	–	2	–	2	–	–	–	–	–	–	2	–	–	–	–
Trifolium subterraneum	–	–	–	–	–	–	–	–	–	–	–	–	1	–	–	–
Vicia faba	–	–	–	–	–	–	–	–	–	–	–	–	–	3	2	–
Vicia villosa	2	1	1	1	1	1	1	1	1	1	1	–	2	1	1	1

FIGURE 5.13 – Classement des espèces issues des références bibliographiques sur le service “fourniture d’azote" . Les colonnes correspondent aux 16 références et les lignes aux espèces. Les numéros dans les cellules correspondent au classement des espèces selon la référence en colonne. Le tiret “–” signifie que l’espèce n’est pas considérée dans la référence.
La référence **(*)** *itab2012* correspond à un ensemble de fiches de l’Institut Technique de l’Agriculture Biologique, voir http://itab.asso.fr/downloads/Fiches-techniques_culture/fiches_especes_engraisverts_2017.pdf.

issues de la bibliographie et de l'outil procèdent en ordre inverse : selon le classement bibliographique, les meilleures espèces ont les notes les plus faibles (elles ont un meilleur rang moyen dans les ordres totaux), alors que selon l'outil c'est l'inverse.

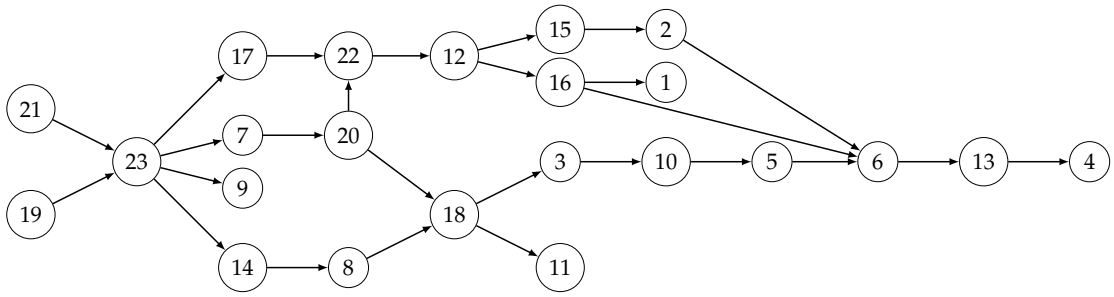
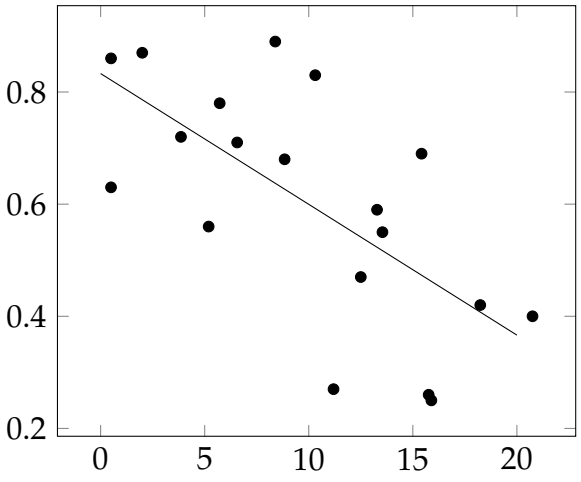


FIGURE 5.14 – Graphe des classements issus de la bibliographie

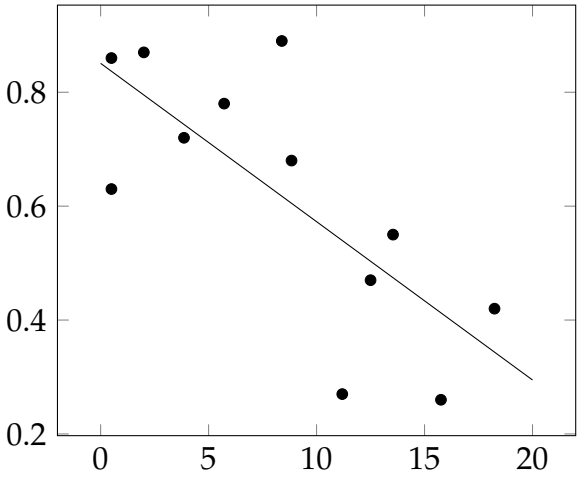
N° Espèce	Nom de l'espèce	Classement biblio	Valeur par l'outil	Fiabilité
1	Avena sativa	18.55	—	—
2	Brassica napus	16.45	0.26	56
3	Dactylis glomerata	11.65	0.27	78
4	Fagopyrum esculentum	21.68	0.40	44
5	Hordeum vulgare	16.58	0.25	44
6	Lolium multiflorum	19.05	0.42	67
7	Lotus Corniculatus	3.95	0.72	67
8	Lupinus albus	6.79	0.71	33
9	Medicago lupulina	12.5	0.47	67
10	Medicago sativa	14.12	0.55	67
11	Melilotus officinalis	16.09	0.69	33
12	Pisum sativum	10.73	0.83	44
13	Raphanus sativus	20.37	—	—
14	Rumex acetosa	4.4	—	—
15	Secale cereale	13.84	0.59	44
16	Sinapsis alba	14.1	—	—
17	Trifolium alexandrinum	5.35	0.56	44
18	Trifolium incarnatum	9.19	0.68	55
19	Trifolium pratense	0.5	0.86	78
20	Trifolium repens	5.9	0.78	78
21	Trifolium subterraneum	0.5	0.63	55
22	Vicia faba	8.71	0.89	56
23	Vicia villosa	2	0.87	55

FIGURE 5.15 – Notation des espèces selon l'étude bibliographique (colonne 3) et l'outil (colonne 4) avec la fiabilité du calcul (colonne 5). Les tirets (—) correspondent à des espèces qui n'apparaissent pas dans la base de faits. En colonne 3, les meilleures espèces ont les notes les plus faibles, et c'est l'inverse en colonne 4.

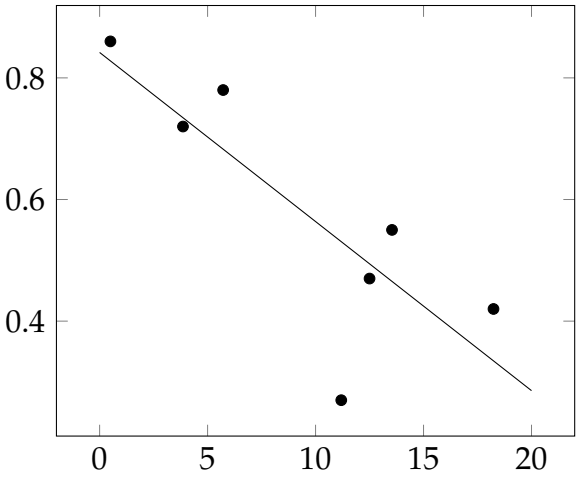
Enfin, nous avons étudié la corrélation entre les scores de l'outil et de la bibliographie, selon le coefficient de corrélation de Pearson r. Dans notre cas, une corrélation parfaite est reflétée par $r = -1$ (puisque l'outil et la bibliographie



(a) Toutes fiabilités



(b) Fiabilité ≥ 50



(c) Fiabilité ≥ 60

FIGURE 5.16 – Corrélation

Valeur	Altitude	Longitude	Latitude	Année
0.0872	1308.23	10.6	46.64	2015
0.3	1476.46	10.57	46.68	2015
1	989.8	10.59	46.66	2015

FIGURE 5.17 – Trois mesures distinctes du trait 403 pour l'espèce Dactylis Glomerata et leurs éléments de contexte

classent les espèces dans des ordres opposés), une corrélation inverse parfaite par $r = 1$, et une absence de corrélation par $r = 0$. Lorsque $r \leq -0.5$, la corrélation est considérée comme bonne.

Dans les graphiques de la figure 5.16, l'axe x correspond aux classements de la bibliographie et l'axe y aux valeurs calculées par l'outil ; chaque point désigne une espèce ; la ligne est une ligne de régression linéaire. En considérant l'ensemble des espèces (Courbe 5.16a), le coefficient de corrélation est déjà significatif ($r = -0.67$).

Nous observons par ailleurs que la fiabilité du score de l'outil est un paramètre crucial. En effet, le coefficient de corrélation s'élève à -0.74 lorsque l'ensemble des espèces est restreint à celles dont la valeur calculée a une fiabilité d'au moins 50% (graphique 5.16b) et à -0.81 pour une fiabilité d'au moins 60% (graphique 5.16c). Dans le cas de ce service, une fiabilité supérieure ou égale à 60% pour une espèce signifie qu'au moins 6 des 9 traits ont effectivement une valeur par le prédicat `aPourValeurTrait` sur cette espèce. Sur le graphique associé à une fiabilité au moins égale à 60% (graphique 5.16c), on peut voir qu'une seule espèce ("dactylis glomerata") est assez éloignée de la droite de régression. Dans le cas de cette espèce en particulier, nous notons une variation importante entre les valeurs de certains traits fonctionnels intervenant dans le rendu du service. Cette variation peut être parfois justifiée par des informations contextuelles différentes entre les mesures entrainant un écart important entre les valeurs. Mais, dans certains cas, la variation entre les valeurs est significative sans qu'aucune information de contexte présente dans la base ne puisse expliquer cet écart.

Par exemple, la figure 5.17 nous donne quelques valeurs du trait ID 403 (ayant pour nom "Plant biomass and allometry : Shoot dry mass (plant aboveground dry mass) per plant" dans TRY) pour l'espèce "Dactylis glomerata". Ces valeurs ont toutes la même unité, elles sont toutes les 3 des types de valeur 'single' et sont prises dans des conditions naturelles de croissance. Les éléments de contexte qui y sont renseignés sont la zone géographique à travers l'altitude, la longitude et la latitude, ainsi que l'année où la mesure a été faite. Nous constatons que

les informations contextuelles pour ces 3 mesures sont sensiblement les mêmes. En effet, elles sont toutes les 3 prises la même année (2015) et dans une même zone géographique, avec une légère différence dans l'altitude. Cette différence d'altitude, ne permet pas de justifier l'écart des valeurs puisqu'on ne trouve pas une corrélation évidente entre l'altitude et la valeur. En effet, la valeur la plus basse est à une altitude de 1308, ensuite, à une altitude un peu plus élevée (1476), la valeur augmente mais, la valeur la plus élevée est à l'altitude la plus basse (989). Finalement, nous notons une différence jusqu'à 11 fois plus entre les valeurs (0.0862 et 1) qu'on ne peut justifier par les données disponibles.

5.4 Discussion

De par l'évaluation des résultats de l'outil, nous constatons que les résultats sont prometteurs. En effet, le classement des espèces proposé par l'outil est le plus souvent similaire au classement issu de la bibliographie. Notons également qu'il y a une corrélation positive entre la fiabilité des valeurs et la similarité entre les deux classements. En effet, plus on se restreint à des espèces ayant des valeurs de fiabilité élevées, plus les classements outil-bibliographie sont proches.

Néanmoins, nous remarquons que pour certaines espèces, le classement de l'outil n'est toujours pas proche de celui de la bibliographie malgré une fiabilité élevée. Nous pensons donc à plusieurs pistes en vue d'améliorer les résultats :

- la prise en compte du contexte, c'est-à-dire, faire en sorte que les mesures prises en considération sont dans un même contexte, ou un contexte proche de celui de l'utilisateur, pour avoir des résultats plus précis. L'intégration des éléments de contexte est un point crucial, puisque, dans nos bases de données actuelles, nous considérons tout type de contexte. Nous pouvons donc avoir des mesures issues d'espèces mesurées dans différents types de climats, de régions, à différentes altitude et même avec des contextes de sol différents. Et nous avons constaté que ces éléments de contexte peuvent fournir des valeurs de traits très différentes pour une même espèce. Par exemple, la figure 5.18 donne les valeurs (0.057 et 13.975) de deux mesures du trait ID 403 dans TRY (ayant pour nom "Plant biomass and allometry : Shoot dry mass (plant aboveground dry mass) per plant") faites sur l'espèce *trifolium pratense*. On remarque que les informations contextuelles sont pratiquement toutes les mêmes (les 2 mesures sont faites dans la même zone géographique, à la même date sur une même espèce à un stade mature, étant en bonne santé, dans des conditions na-

turelles et dans un même habitat). Les seules informations qui diffèrent considérablement sont les teneurs en azote et phosphore du sol. En se basant sur les informations récupérées, cette variation de valeur peut être causée par cette différence en teneur.
Dans notre cas, nous n'avons pas pris en compte les éléments de contexte, principalement pour trois raisons : (a) même en considérant tout type de contexte, nous avons un problème de valeurs manquantes, supprimer certaines valeurs n'était donc pas envisageable ; (b) les informations de contexte ne sont pas toutes suffisamment renseignées dans les bases utilisées, ou alors, difficilement exploitables ; et (c) les connaissances agronomiques de l'impact du contexte sur les mesures de traits fonctionnels sont encore en développement et instables, il peut donc être difficile de déterminer exactement quels éléments de contexte ont un impact sur les différents traits.

- le choix d'une méthode d'agrégation différente pour le passage des traits aux fonctions puis aux services et/ou la modification de la pondération pour donner plus d'importance à un trait ou à une fonction dans le rendu du service. En effet, pour l'instant, nous avons accordé une importance égale à tous les traits et toutes les fonctions puis nous avons choisi comme méthode d'agrégation la moyenne puisqu'il n'y a pas suffisamment d'études disponibles permettant de nous guider dans le choix d'une meilleure méthode d'agrégation.

Cependant, comme nous avons pu le constater avec le cas de l'espèce *dactylis glomerata*, la prise en compte du contexte n'assure pas d'avoir à tous les coups un bon classement des espèces par l'outil. Dans ce cas, on pourrait soit supposer qu'il y'ait d'autres éléments de contexte qui expliqueraient ces différences mais qui ne sont pas indiqués dans les sources de données ; soit mettre en cause la fiabilité des observations.

ID de l'observation	**2507009**	**2507138**
Valeur	0.057	13.975
Latitude	58.42	58.66
Longitude	22.02	23.21
Altitude	17.5	19
Mesure	8/1/2010	8/1/2010
Exposition	natural environment	natural environment
Habitat	calcareous grassland	calcareous grassland
Maturité	mature	mature
État de santé	healthy	healthy
Profondeur du sol	10.6 cm	7.57 cm
Ph du sol	6.8	7.04
Teneur en azote du sol	0.38 %	0.88 %
Teneur en phosphore du sol	1.41 mg/kg	21.53 mg/kg

FIGURE 5.18 – Exemple de variation des valeurs du trait 403 pour l'espèce Trifolium pratense mesurées dans des contextes (de sol) différents (ObservationID : 2507009 et 2507138).

CHAPITRE 6

Explications

Ce chapitre présente un mécanisme permettant de justifier la présence d'un fait de la base de faits saturée en fournissant à l'utilisateur les informations (faits, règles appliquées, homomorphismes utilisés) pertinentes ayant permis la déduction de ce fait. Ces informations sont structurées sous la forme d'*explications*. Une explication *pertinente* pour un utilisateur doit être exprimée en se limitant au vocabulaire qui a du sens pour lui, typiquement celui de l'ontologie de domaine.

Ce mécanisme d'explication a plusieurs intérêts. Tout d'abord, il rend le raisonnement transparent à l'utilisateur en lui permettant de comprendre les différentes étapes de la production d'un fait. Cette compréhension du raisonnement permet à un utilisateur expert d'analyser et d'affiner les résultats de l'outil, par exemple en modifiant les fonctions d'agrégation permettant de calculer les différentes valeurs de fonction et de service. D'autre part, un utilisateur final pourra identifier dans l'explication des faits qu'il juge incorrects et les corriger éventuellement grâce au mécanisme de *prise en compte des retours utilisateurs* (voir la section 5.1.5). Par exemple, si la valeur d'un service pour une espèce donnée est inattendue pour l'utilisateur, le mécanisme d'explication peut fournir les valeurs des fonctions reliées à ce service pour cette même espèce, ainsi que la règle de calcul. Dès lors, l'utilisateur pourra identifier parmi ces valeurs de fonctions celles qui lui semble incorrectes et les corriger, ou bien remonter plus loin au niveau des valeurs de traits ayant conduit à ces valeurs de fonction et les corriger.

Ce chapitre est divisé en 3 sections :

- En section 6.1, nous présentons le *graphe de dépendance des atomes* : une

structure construite au cours de la dérivation qui permet de garder trace des applications de règles.
- En section 6.2, nous présentons notre cadre explicatif dans le cas de règles Datalog de base.
- En section 6.3, nous étendons notre cadre explicatif en considérant la négation et les fonctions calculées.

6.1 Graphe de dépendance des atomes

Dans cette section nous présentons le *graphe de dépendance des atomes* qui encode une dérivation. Pour cela, nous aurons besoin de quelques définitions préliminaires.

Définition 6.1.1 (Hypergraphe orienté) *Un* hypergraphe orienté $\mathcal{H}$ *est un couple* (V, E) *tel que* V *est un ensemble de noeuds et* E *un ensemble d'*hyper-arcs. *Chaque hyper-arc est une paire* $e = (V_1, V_2)$, *où* V_1 *et* V_2 *sont deux sous-ensembles de* V *non vides et disjoints ;* V_1 *est appelé* origine *de* e *et* V_2 extrémité *de* e.

Définition 6.1.2 (Graphe biparti orienté) *Un* graphe biparti orienté *est une structure* $G = (V_1, V_2, E)$ *où* V_1 *et* V_2 *sont deux ensembles disjoints et non vides dont l'union forme l'ensemble des sommets de* G *et* E *est l'ensemble des arcs* (x, y) *tels que si* $x \in V_1$ *alors* $y \in V_2$ *et si* $x \in V_2$ *alors* $y \in V_1$.

À un hypergraphe $\mathcal{H} = (V, E_{\mathcal{H}})$, on peut faire correspondre un graphe biparti orienté, où les hyper-arcs deviennent des sommets. Plus précisément, $\mathcal{H}$ correspond au graphe biparti orienté $G = (V_1, V_2, E_G)$ où
- $V_1 = V$;
- $V_2 = E_{\mathcal{H}}$;
- E_G : pour tout hyper-arc $e \in E_{\mathcal{H}}$, de la forme $e = (V_{1_{\mathcal{H}}}, V_{2_{\mathcal{H}}})$: il y a un arc (v_1, e) pour tout $v_1 \in V_{1_{\mathcal{H}}}$ et un arc (e, v_2) pour tout $v_2 \in V_{2_{\mathcal{H}}}$.

On rappelle qu'une dérivation de la base de connaissances $\mathcal{K} = (F, \mathcal{R})$, à partir de la base de faits initiale F, est une suite $(F_0 = F), F_1, \ldots, F_n$ telle que F_n est le *résultat de la dérivation* et pour tout $0 < i \leq n$, F_i est obtenu de F_{i-1} en appliquant une règle de $\mathcal{R}$: il existe une règle $(R) : C \to T$ et une réponse (un homomorphisme) π à C dans F_{i-1} avec $F_i = \alpha(F_{i-1}, R, \pi)$. Chaque (R, π) est appelé un *trigger* (sur la base de faits F_{i-1}). On peut encoder une dérivation D sous la forme d'un hypergraphe $\mathcal{H} = (V, E_{\mathcal{H}})$ tel que $V = F_n$ et $E_{\mathcal{H}}$ est en bijection avec l'ensemble des triggers de D. Nous définissons ci-dessous le graphe

de dépendance des atomes d'une dérivation, qui correspond au graphe biparti orienté associé à cet hypergraphe.

Définition 6.1.3 (Graphe de dépendance des atomes d'une dérivation) *Le* graphe de dépendance des atomes *(en abrégé* GAD*) associé à une dérivation* $D = (F_0 = F), F_1, \ldots, F_n$ *est le graphe biparti orienté* $G(D) = (V_1, V_2, E_G)$ *tel que :*

- $V_1 = F_n$;
- $V_2 = \{t_1, ..., t_n\}$ *où chaque* t_i *est un trigger de* D ;
- *pour tout trigger* $t = (R, \pi)$ *de* D*, il y a un arc de chaque atome* α *de* $\pi(corps(R))$ *vers* t *et un arc de* t *vers chaque atome* α' *de* $\pi(tête(R))$.

Exemple 6.1.1 *Soit* $\mathcal{K} = (F, \mathcal{R})$ *une base de connaissances telle que :*

$F = \{t(a,b), t(a,c), q(b), q(a)\}$
$\mathcal{R} = \{R_1, R_2, R_3, R_4, R_5\}$
$R_1 = t(X,Y), q(Y) \rightarrow r(X,Y)$
$R_2 = t(X,Y), q(X) \rightarrow v(X,Y)$
$R_3 = r(X,Y), v(X,Z) \rightarrow s(Y,Z)$
$R_4 = v(X,Y), q(X) \rightarrow s(X,Y)$
$R_5 = s(X,Y) \rightarrow p(Y)$

On considère la dérivation complète $D_1 = F_0,\ F_1,\ F_2,\ F_3,\ F_4,\ F_5$ *définie par la relation de récurrence suivante :*

$$\begin{cases} F_0 &= F \\ F_i &= F_{i-1} \cup \pi_i(tête(R_i)),\ 0 < i \leq 5 \end{cases}$$

avec :

$\pi_1 = \{(X,a), (Y,b)\}$
$\pi_2 = \{(X,a), (Y,c)\}$
$\pi_3 = \{(X,a), (Y,b), (Z,c)\}$
$\pi_4 = \pi_2$
$\pi_5 = \{(X,b), (Y,c)\}$

La figure 6.1 (sans la partie grisée) représente le graphe de dépendance des atomes associé à la dérivation D_1, qui produit la base de faits saturée

$$F_5 = \{t(a,b), q(b), t(a,c), q(a), r(a,b), v(a,c), s(b,c), s(a,c), p(c)\}$$

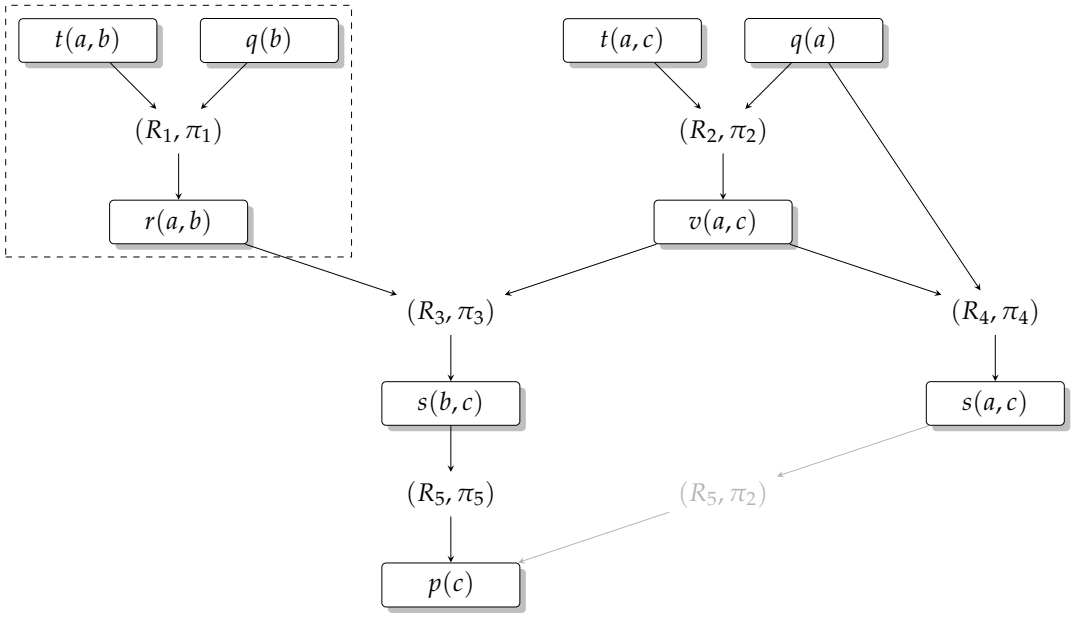


FIGURE 6.1 – Graphe de dépendance des atomes d'une dérivation D_1 de la base de connaissances $\mathcal{K}$.

Les noeuds correspondants aux faits sont représentés par des rectangles, les autres noeuds (non encadrés) représentent les triggers. Le graphe se lit comme suit, prenons par exemple la partie encadrée en pointillés : "le fait $r(a,b)$ a été produit en appliquant la règle R_1 sur les faits $t(a,b)$ et $q(b)$, et en utilisant l'homomorphisme π_1."

Comme nous le constatons, ce graphe contient la trace des raisonnements (relatifs à une dérivation donnée, ici D_1). En effet, pour tout atome de la base de faits saturée, on peut déterminer l'ensemble des faits initiaux et une séquence d'applications de triggers qui ont permis de produire cet atome. Si nous reprenons l'exemple précédent, nous voyons que l'atome $p(c)$ a été déduit par l'ensemble des faits initiaux suivants $\{t(a,b), q(b), t(a,c), q(a)\}$ en appliquant par exemple la séquence de triggers $\Big[(R_1,\pi_1),(R_2,\pi_2),(R_3,\pi_3),(R_5,\pi_5)\Big]$. Remarquons qu'on peut associer plusieurs dérivations à un même graphe, car les triggers qui ne sont pas sur un même chemin (comme (R_1,π_1) et (R_2,π_2)) peuvent s'ordonner de différentes façons.

Notons également que pour une même base de faits saturée, le graphe de dépendance des atomes est susceptible de changer suivant la dérivation (complète) considérée. En effet, en reprenant le même exemple, une autre dérivation D_2 de $\mathcal{K}$ peut produire $p(c)$ par une application du trigger (R_5,π_2) sur $s(a,c)$. Le graphe correspondant s'obtient à partir de celui de la dérivation D_1 (figure 6.1) en supprimant le trigger (R_5,π_5) et en le remplaçant par celui

Par ailleurs, nous aurions pu également considérer une dérivation D qui produit $p(c)$ de deux façons différentes. Dans ce cas, le graphe associé correspondra au graphe en figure 6.1, partie grisée incluse. Cependant, cette dérivation comportera une application inutile puisque $p(c)$ est produit deux fois. Nous considérons donc par la suite uniquement des dérivations ne contenant que des applications utiles.

6.2 Explications pertinentes

Dans cette section, nous allons utiliser le GAD pour produire des explications. Pour cela, nous divisons notre section en trois parties : la première définit le graphe des ancêtres d'un atome qui correspond à un GAD restreint aux ancêtres de l'atome à expliquer ; la seconde explique pourquoi deux notions d'explications classiques ne nous paraissent pas satisfaisantes ; la troisième introduit notre proposition, celles des explications pertinentes.

6.2.1 Graphe des ancêtres associé à un GAD

Définition 6.2.1 (Ancêtre d'un atome dans un GAD) *Soient $\mathcal{K} = (F, \mathcal{R})$ une base de connaissances, $D = F_0, \ldots, F_n$ une dérivation de $\mathcal{K}$ et $G(D)$ le graphe de dépendance des atomes associé à la dérivation D. Un* ancêtre *d'un atome α dans $G(D)$ est un atome α' de $G(D)$ tel que il existe un chemin allant de α' à α.*
α' est un ancêtre direct *de α dans $G(D)$ si α est produit à partir de α' (et possiblement d'autres atomes) par l'application d'un seul trigger.*
α' est un ancêtre initial *de α dans $G(D)$ si α' est un ancêtre de α et $\alpha' \in F_0$.*

Dans l'exemple 6.1.1, en considérant le graphe $G(D_1)$ associé à la dérivation D_1 (c'est-à-dire sans la partie grisée), on a : $s(b, c)$ est un ancêtre direct de $p(c)$, $r(a, b)$ est un ancêtre de $p(c)$; et $t(a, b)$ est un ancêtre initial de $p(c)$, mais $s(a, c)$ n'est pas un ancêtre de $p(c)$ (car il n'existe aucun chemin allant de $s(a, c)$ à $p(c)$).

Par la suite, pour expliquer un atome, nous ne nous intéresserons qu'à ses ancêtres dans un GAD donné. Nous définissons donc le *graphe des ancêtres* d'un atome α qui est obtenu à partir du GAD en retirant les noeuds (faits) qui ne sont pas des ancêtres de α, les triggers qui produisent ces noeuds et les arcs qui leur sont incidents. En gardant le même exemple 6.1.1, le graphe des ancêtres de l'atome $p(c)$ est dessiné en figure 6.2.

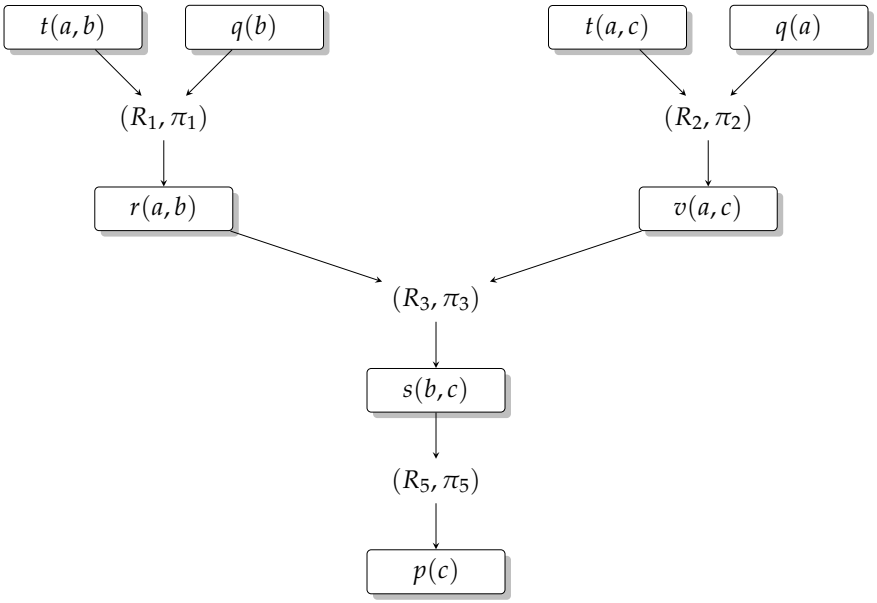


FIGURE 6.2 – Graphe des ancêtres de l'atome $p(c)$ associé au GAD de l'exemple 6.1.1.

6.2.2 À la recherche d'une notion d'explication satisfaisante

On peut considérer deux façons d'expliquer un atome α qui sont naturellement associées à la notion de dérivation. (i) Soit on donne un ensemble de faits initiaux tel qu'il existe une dérivation à partir de ces faits permettant de produire α. L'explication consiste en cet ensemble d'atomes, qu'on veut minimal au sens de l'inclusion : soit minimal pour l'ensemble des dérivations possibles, soit minimal étant donné une dérivation déjà choisie et dans ce cas, l'explication est définie par l'ensemble des ancêtres initiaux de α dans cette dérivation. (ii) Soit, étant donné une dérivation, on donne un trigger (R,π) de cette dérivation qui produit α ainsi que l'ensemble de faits $\pi(corps(R))$. Ce type d'explication sera appelé explication *directe* de α. Cependant, aucune de ces deux méthodes ne nous parait adaptée à notre objectif, ce que nous allons illustrer sur notre cadre d'étude.

(i) Donner les faits provenant de la base de faits initiale pour expliquer un fait déduit n'est pas très informatif : on ne sait rien des règles qui interviennent dans la dérivation. Par exemple, dans notre cas d'étude, si on cherchait une explication initiale de *aPourValeurService(service, nom de l'espèce, valeur, fiabilité)*, on se retrouverait avec des explications contenant entre autres des faits de la forme suivante :

- *aPourValeurBase(ID du trait, ID de l'espèce, growing condition, valeur, base)*,
- *aPourEspeceID(nom de l'espèce, ID de l'espèce, base)*,
- *estLieATraitID(nom du trait, ID du trait, base)*,

- *basePrioritaire(base1, base2)*,
- *aPourID(nom du trait, ID du trait, base)*,
- etc.

Cependant, nous n'aurions pas les informations pertinentes souhaitées comme : les valeurs des traits "experts" des diagrammes, les valeurs des fonctions, la façon dont la valeur de service a été calculée, etc. Nous détaillons ci-dessous ce que l'on obtiendrait pour l'atome *aPourValeurService("nitrogen supply to the vine", "dactylis glomerata", 0.27, 78, "n")*.

Exemple 6.2.1 *Supposons qu'on veuille une explication "initiale" de la valeur du service* Nitrogen supply to the vine *de l'espèce* dactylis glomerata *dans des conditions de croissance naturelles, c'est-à-dire une explication initiale du fait :* aPourValeurService("nitrogen supply to the vine", "dactylis glomerata", 0.27, 78, "n"). *Les faits qui composent l'explication sont les suivants :*

```
% Valeurs des traitIDs présents dans les bases
aPourValeurTraitInitiale(3117, 16700, n, 0.3, try).
aPourValeurTraitInitiale(3116, 16700, n, 0.5, try).
aPourValeurTraitInitiale(3115, 16700, n, 0.2, try).
aPourValeurTraitInitiale(403, 16700, n, 0.1, try).
aPourValeurTraitInitiale(700, 16700, n, 0.2, try).
aPourValeurTraitInitiale(47, 16700, n, 0.5, try).
aPourValeurTraitInitiale(146, 16700, n, 0.1, try).
aPourValeurTraitInitiale(614, 16700, n, 0.4, try).
aPourValeurTraitInitiale(1080, 16700, n, 1, try).
aPourValeurTraitInitiale(2025, 16700, n, 0.4, try).
aPourValeurTraitInitiale(8, 16700, n, 0, try).

% Correspondances IDs - traits experts
aPourTraitID("specific leaf area",3117, try).
aPourTraitID("specific leaf area",3116, try).
aPourTraitID("specific leaf area",3115, try).
aPourTraitID("dry mass of plants, shoots, leaves",403, try).
aPourTraitID("dry mass of plants, shoots, leaves",700, try).
aPourTraitID("dry matter content of leaves",47, try).
aPourTraitID("C/N ratio of the plant, root, leaf, litter", 146,
try).
aPourTraitID("specific root length", 614, try).
aPourTraitID("specific root length", 1080, try).
aPourTraitID("root length density",2025, try).
aPourTraitID("nitrogen fixation capacity", 8, try).

% Liens fonctions - service "Nitrogen supply to the vine"
estPositivementLieAFonction("specific leaf area",
```

```
"mineralization of organic matter").
estPositivementLieAFonction("dry mass of plants, shoots, leaves",
"mineralization of organic matter").
estNegativementLieAFonction("dry matter content of leaves",
"mineralization of organic matter").
estNegativementLieAFonction("C/N ratio of the plant, root, leaf,
litter", "mineralization of organic matter").
estPositivementLieAFonction("specific root length","soil
exploration and competition with vines").
estPositivementLieAFonction("root length density","soil
exploration and competition with vines").
estPositivementLieAFonction("nitrogen fixation capacity",
"symbiotic fixation of atmospheric nitrogen").
estPositivementLieAService("mineralization of organic matter",
"nitrogen supply to the vine").
estNegativementLieAService("soil exploration and competition
with vines", "nitrogen supply to the vine").
estPositivementLieAService("symbiotic fixation of atmospheric
nitrogen", "nitrogen supply to the vine").

% Agrégations pour les passages traits-fonctions-service
aPourMethodeAgregation("mineralization of organic matter",
"mean").
aPourMethodeAgregation("soil exploration and competition with
vines", "mean").
aPourMethodeAgregation("symbiotic fixation of atmospheric
nitrogen", "mean").
aPourMethodeAgregation("nitrogen supply to the vine", "mean").


% Autres informations
aPourEspeceID("dactylis glomerata", 16700, try).
basePrioritaire(try,db2).
```

Malgré certaines informations pertinentes comme les différents liens entre traits, fonctions et service, l'ensemble des informations ne permet pas vraiment d'expliquer comment la valeur de service a été obtenue. Par exemple, on a les valeurs des IDs de traits dans les bases de données mais pas les valeurs des traits experts, ni des fonctions.

(*ii*) Par ailleurs, donner une succession d'explications directes peut devenir très *long* et noyer l'utilisateur sous des détails de calcul ; en outre, les prédicats utilisés ne sont pas forcément intelligibles pour l'utilisateur (même si on imaginait une traduction de ces explications en langue naturelle). On voudrait se restreindre à des explications moins nombreuses qui aient du sens pour l'utilisateur.

Exemple 6.2.2 *Reprenons l'exemple précédent mais cherchons cette fois une explication directe du fait* aPourValeurService("nitrogen supply to the vine", "dactylis glomerata", 0.27, 78, "n"). *Ici aussi pour simplifier, nous ne donnerons pas les triggers explicatifs mais juste l'ensemble des faits. Nous aurons :*

```
% Étape 1 :
aPourValeurServiceCalculee("nitrogen supply to the vine",
"dactylis glomerata", 0.27, 78, "n").
```

L'étape 1 nous donne une explication directe du fait souhaité. Cette explication apporte le fait que la valeur de service a été calculée (et non pas donnée directement par une source) mais cela n'aide pas beaucoup l'utilisateur. Il peut donc continuer en demandant une explication du fait produit à l'étape 1 :

```
% Étape 2 :
estLieA3Fonctions("nitrogen supply to the vine", "mineralization
of organic matter", 1, "soil exploration and competition with
vines", -1, "symbiotic fixation of atmospheric nitrogen" ,1,
mean).
aPourValeurFonction("mineralization of organic matter", "dactylis
glomerata", 0.5, 80, "n").
aPourValeurFonction("soil exploration and competition with vines",
"dactylis glomerata", 0.7, 66, "n").
aPourValeurFonction("symbiotic fixation of atmospheric nitrogen",
"dactylis glomerata", 0, 100, "n").
nbrTrait("mineralization of organic matter",5).
nbrTrait("soil exploration and competition with vines",3).
nbrTrait("symbiotic fixation of atmospheric nitrogen",1).
```

Cette fois, nous avons les valeurs des fonctions qui ont permis de calculer la valeur de service. Cependant, on fournit à l'utilisateur un vocabulaire auquel il n'est pas habitué, et qui ne fait pas forcément sens pour lui, comme le prédicat `estLieA3Fonctions` *qui n'est pas dans l'ontologie de domaine. A nouveau, si cette explication n'est pas suffisante, l'utilisateur peut demander une explication directe d'un fait produit à l'étape 2. Ici, nous cherchons une explication du fait :*

```
estLieA3Fonctions("nitrogen supply to the vine", "mineralization
of organic matter", 1, "soil exploration and competition with
vines", -1, "symbiotic fixation of atmospheric nitrogen" ,1,
mean)
```

L'explication est donnée à l'étape 3 ci-dessous :

```
 % Étape 3 :
fonctionMax3("nitrogen supply to the vine", "mineralization of
```

```
organic matter", 1, "soil exploration and competition with vines",
-1, "symbiotic fixation of atmospheric nitrogen" ,1).
aPourMethodeAgregation("nitrogen supply to the vine", "mean").
aPourValeurFonctionCalculee("mineralization of organic matter",
"dactylis glomerata", 0.5, 80, "n").
aPourValeurFonctionCalculee("soil exploration and competition
with vines", "dactylis glomerata", 0.7, 66, "n").
aPourValeurFonctionCalculee("symbiotic fixation of atmospheric
nitrogen", "dactylis glomerata", 0, 100, "n").

 % L'étape 3 inclut aussi des faits ayant pour prédicats :
estLieA5Traits, estLieA3Traits et estLieA1Trait, dont nous ne
donnerons pas la liste exhaustive ici. Ces faits ont permis
la production des 3 derniers faits de l'étape 2
```

Nous voyons une fois de plus que l'étape 3 ne fournit aucune explication pertinente et peut "embrouiller" l'utilisateur avec de nouveaux prédicats non familiers : `fonctionMax3, aPourValeurFonctionCalculee, estLieA5Traits, etc.`

Notre objectif est donc de définir une notion d'explication intermédiaire entre la notion d'explication par des atomes de la base de faits, trop peu informative sur les déductions faites, et la notion d'explication comme une succession d'explications directes, trop détaillée. Cette notion d'explication sera définie par rapport à une dérivation en particulier, nous aurons donc besoin de la définition suivante :

Définition 6.2.2 (Explication d'un atome suivant une dérivation) *Soient $\mathcal{K}$ une base de connaissances et $G(D)$ le GAD associé à la dérivation D de $\mathcal{K}$. Une* explication *d'un atome α suivant D est un couple $exp_D(\alpha) = (A, T)$ tel que A est un ensemble d'ancêtres de α dans G(D), T est un trigger (R, π), $A = \pi(body(R))$ et $\alpha \in \pi(head(R))$.*

Notons que dans la définition précédente, R n'est pas forcément une règle de la base de connaissance $\mathcal{K}$, mais plus généralement une règle qui se déduit de $\mathcal{K}$. Par exemple, on pourrait prendre la règle $A \rightarrow \alpha$, mais elle ne nous intéresse pas beaucoup car elle n'apporte rien. On va plutôt considérer soit des règles de $\mathcal{K}$ (ce qu'on appelera explication directe), soit des règles obtenues en composant des règles de $\mathcal{K}$.

6.2.3 Notre proposition

Nous proposons de considérer un sous-ensemble des prédicats du vocabulaire de la base de connaissance, appelé ensemble des *prédicats pertinents* (dans notre cas d'étude, ce serait typiquement ceux intervenant dans l'ontologie) et

de donner des explications utilisant uniquement ce vocabulaire. Intuitivement, ceci revient à restreindre le graphe des ancêtres (d'un fait à expliquer) aux faits construits sur des prédicats pertinents, qu'on appelle *faits pertinents*. Il faut donc associer à ce *graphe des ancêtres pertinents* de nouvelles règles obtenues en *composant* les règles d'origine, de façon à faire disparaitre les prédicats non pertinents. Nous commençons par rappeler quelques définitions de logique.

Définition 6.2.3 (Composition de substitutions) *Soient s_1 et s_2 deux substitutions telles que :*

$$s_1 = \{(X_1, t_1), \cdots, (X_k, t_k)\}$$

$$s_2 = \{(Y_1, t'_1), \cdots, (Y_k, t'_k)\}$$

La composition *de s_2 et s_1 est (on applique s_1 puis s_2) :*

$$s_2 \circ s_1 = \{(X_i, s_2(t_i)) | X_i \neq s_2(t_i)\} \cup \{(Y_j, t'_j) | \forall i, Y_j \neq X_i\}$$

Définition 6.2.4 (Unificateur de termes) *Soient $l_1, \cdots, l_n$ des listes de termes. Ces listes sont* unifiables *s'il existe une substitution s qui les rend identiques :*
$s(l_1) = \cdots = s(l_n)$. *s est appelé* unificateur *de ces listes de termes.*

Définition 6.2.5 (Unification d'atomes) *Des atomes sont* unifiables, *s'ils ont le même prédicat et si leurs listes de termes sont unifiables.*

Exemple 6.2.3 *Soit l'ensemble d'atomes $E = \{p(X, b, Z), p(Y, W, a), p(Y, b, Z)\}$. L'unificateur $u = \{(W, b), (X, c), (Y, c), (Z, a)\}$ unifie ces deux atomes en $p(c, b, a)$.*

Définition 6.2.6 (Unificateur le plus général) *Un unificateur u d'un ensemble E est un* unificateur le plus général *(en abrégé* upg*) si tout autre unificateur u' de E s'obtient par une substitution supplémentaire s : $u'(E) = s(u(E)) = (s \circ u)(E)$.*

Dans notre exemple précédent l'unificateur $u = \{(W, b), (X, c), (Y, c), (Z, a)\}$ n'est pas un upg de E. En effet, il existe un unificateur $u' = \{(W, b), (X, Y), (Z, a)\}$ tel que $u(E) = s(u'(E))$ pour $s = \{(Y, c)\}$.

L'ensemble E peut avoir plusieurs upg. Dans ce cas, ils s'obtiennent les uns des autres par un renomage bijectif des variables.

Par la suite, nous n'avons besoin d'unifier que deux atomes à la fois, au lieu d'un ensemble E de cardinalité quelconque.

Définition 6.2.7 (Composition de règles) *R_1 est* composable *avec R_2 s'il existe un unificateur u, tel que $u(tête(R_1)) \subseteq u(corps(R_2))$*

La composé de deux règles R_1 et R_2 suivant un unificateur u est notée $R_2 \circ_u R_1$. Par la suite, lorsqu'il n'y a pas d'ambiguïté on utilisera simplement la notation $R_2 \circ R_1$.

Exemple 6.2.4 *Reprenons les règles suivantes (exemple 6.1.1) :*

$$R_1 : t(X_1, Y_1), q(Y_1) \rightarrow r(X_1, Y_1)$$
$$R_3 : r(X_3, Y_3), v(X_3, Z_3) \rightarrow s(Y_3, Z_3)$$

Soit la substitution $u = \{(X_3, X_1), (Y_3, Y_1)\}$. *On a* $u(tête(R_1)) = r(X_1, Y_1) = u(r(X_3, Y_3))$. *C'est donc un unificateur qui montre que* R_1 *est composable avec* R_3. *Autrement dit, on a* $u(tête(R_1)) \subseteq u(corps(R_3)) = \{r(X_1, Y_1), v(X_1, Z_3)\}$. *On obtient :*

$$R_3 \circ R_1 : t(X_1, Y_1), q(Y_1), v(X_1, Z_3) \rightarrow s(Y_1, Z_3)$$

Précisons que dans notre cas il existera toujours un unificateur pour les règles R_1 et R_2 qu'on cherchera à composer car on sera guidé par une dérivation qui aura appliqué R_2 en utilisant un atome produit par l'application de R_1.

Dans la suite, on note $\mathcal{V}$ *l'ensemble des prédicats pertinents* et on impose que $\mathcal{V}$ contienne *tous* les prédicats qui apparaissent dans la base de faits (initiale).

Définition 6.2.8 (Atome pertinent (par rapport à un vocabulaire)) *Un atome* $\alpha = p(t_1, \ldots, t_k)$ *est* pertinent *par rapport à* $\mathcal{V}$ *si* $p \in \mathcal{V}$.

Un ancêtre α' de α est pertinent (par rapport à un vocabulaire) si α' est un atome pertinent.

Définition 6.2.9 (Explication pertinente d'un atome suivant un vocabulaire) *Une explication* (A, T) *d'un atome* α *est* pertinente *si* A *n'est formée que d'atomes pertinents, ou bien de façon équivalente, telle que le corps de la règle* R *ne contient que des atomes pertinents.*

Remarquons que, dans la définition précédente, α est lui-même censé être un atome pertinent. Sinon cela n'a pas de sens que l'utilisateur en demande une explication.

Par la suite, pour transformer une explication (A, T) non pertinente en une explication pertinente, on va itérativement composer la règle de T avec les règles qui produisent les atomes de A qui sont non pertinents, jusqu'à n'avoir dans le corps de la règle que des atomes pertinents. Ce processus permet de remplacer tous les atomes non pertinents puisque les faits de la base de faits initiale sont

par définition pertinents. Plus précisément, pour construire une explication pertinente de α on part de son explication directe $(A, (R, \pi))$ et on répète l'action suivante tant que l'explication obtenue n'est pas pertinente : soit a_i un atome non pertinent de A et soit α_i un atome du corps de R qui produit a_i, c'est-à-dire tel que $a_i = \pi(\alpha_i)$; soit $(A_i, (R_i, \pi_i))$ l'explication directe de a_i ; soit u_i l'upg de α_i et de tête(R_i) ; on construit l'explication $(A', (R', \pi'))$ où :

- $R' = R \circ_{\mu_i} R_i$;
- π' est défini ainsi : pour toute variable x de corps(R'), soit une variable y tel que $\mu_i(y) = x$ (éventuellement $x = y$), alors $\pi'(x) = \pi_i(y)$ si y est dans corps(R_i), sinon $\pi'(x) = \pi(y)$;
- $A' = \pi'(\text{corps}(R'))$.

Exemple 6.2.5 *Reprenons le graphe des ancêtres présenté en figure 6.2.*
Le vocabulaire pertinent considéré est $\mathcal{V} = \{v, p, t, q\}$. Nous cherchons une explication pertinente à $p(c)$.
(R_5, π_5) est une explication directe de $p(c)$ avec $\pi_4(corps(R_5)) = \{s(b,c)\}$ et $s(b,c)$ n'est pas un atome pertinent. On remonte donc dans le graphe en cherchant une explication (directe) de $s(b,c)$. L'explication directe de $s(b,c)$ est (R_3, π_3) avec $\pi_3(corps(R_3)) = \{r(a,b), v(a,c)\}$. $v(a,c)$ est un atome pertinent car $v \in \mathcal{V}$ mais $r(a,b)$ n'est pas pertinent. On cherche alors une explication pertinente à $r(a,b)$ en prenant à nouveau une explication directe : (R_1, π_1) avec $\pi_3(corps(R_1)) = \{t(a,b), q(b)\}$. Cette fois $t(a,b)$ et $q(b)$ sont tous les deux des atomes pertinents. On construit donc l'explication pertinente (A, T) de $p(c)$ avec $T = (R, \pi)$ tel que

$$R = R_5 \circ (R_3 \circ R_1) : t(X_3, X_5), q(X_5), v(X_3, Y_5) \rightarrow p(Y_5)$$

$$\pi = \{(X_5, b), (Y_5, c), (X_3, a)\}$$

et $A = \pi(corps(R)) = \{t(a,b), q(b), v(a,c)\}$.

Par la suite, on peut représenter un graphe qui est restreint aux ancêtres pertinents de l'atome qu'on souhaite expliquer.

Exemple 6.2.6 *Si nous prenons l'exemple 6.2.5 précédent, le graphe des ancêtres pertinents associé sera celui présenté en figure 6.3.*

Remarquons que les atomes de A dans l'explication de $p(c)$ produite correspondent aux plus proches ancêtres pertinents de $p(c)$.

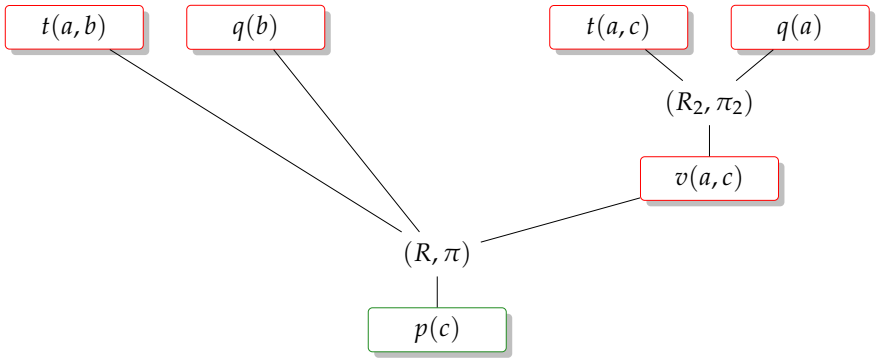


FIGURE 6.3 – Graphe des ancêtres pertinents de l'atome $p(c)$

6.3 Extension aux règles avec négation et fonctions calculées

On considère maintenant que notre ensemble de règles peut contenir des négations d'atomes et des fonctions calculées.

6.3.1 Gestion des fonctions calculées

Concernant les fonctions calculées, la difficulté est qu'après la composition de règles, on peut se retrouver avec des fonctions calculées dans le corps d'une règle, et plus précisément dans un atome standard du corps d'une règle, ce qui n'est pas permis dans notre syntaxe. Une façon de résoudre ce problème est de remplacer chaque fonction f_i présente dans un atome standard du corps d'une règle par une variable Z_i et ajouter un prédicat calculé qui décrit l'égalité entre chaque fonction et la variable qui a pris sa place $f_i = Z_i$.

Exemple 6.3.1 *Prenons par exemple les règles suivantes :*

$$R_1 : p(X) \rightarrow r(f(X,Y))$$

$$R_2 : s(U), r(U) \rightarrow q(U)$$

avec $f(X,Y)$ une fonction calculée.
La composition $R_2 \circ R_1$ suivant l'unification $u = \{(U, f(X,Y))\}$, nous donne

$$R_2 \circ R_1 : p(X), s(f(X,Y)) \rightarrow q(f(X,Y))$$

On sort $f(X,Y)$ de l'atome $s(f(X,Y))$ en le remplaçant par la variable Z et en ajoutant une égalité :

$$R_2 \circ R_1 : p(X), s(Z), Z = f(X,Y) \rightarrow q(Z)$$

Nous obtenons donc des règles dans une syntaxe autorisée par notre cadre formel.

Notons que dans notre cas d'étude la situation est plus simple car, avec les règles actuelles, la composition ne produit jamais de fonctions calculées dans les prédicats standards. En effet, pour que cela arrive il faut que la tête d'une règle R_1 comportant un terme calculé soit unifiée avec une variable X du corps d'une règle R_2 qui apparait ailleurs dans le corps de R_2. Sinon, la réécriture efface le terme calculé. L'exemple 6.3.2 illustre le type de composition de règles la plus fréquente qu'on a dans notre cas d'étude avec la présence de fonctions calculées.

Exemple 6.3.2 *On considère deux règles R_1 et R_2 de notre ensemble de règles. R_1 permet de calculer une valeur de service à partir des valeurs de fonctions qui lui sont reliées. R_2 permet de passer à une valeur finale de service. Notons que R_2 contient un atome nié dans le corps (en gris). On ne va pas le considérer dans cet exemple (pour l'instant il sera ignoré), la gestion de ce type d'atome étant détaillée dans la prochaine section.*

R_1 : estLie3AFonctions(S1,F1,P1,F2,P2,F3,P3,Ag),
aPourValeurFonction(F1,Esp1,V1,Fiab1,GrowCond1),
aPourValeurFonction(F2,Esp1,V2,Fiab2,GrowCond1),
aPourValeurFonction(F3,Esp1,V3,Fiab3,GrowCond1),
nbrTrait(F1,NbTrait1), nbrTrait(F2,NbTrait2),
nbrTrait(F3,NbTrait3)
→ **aPourValeurServiceCalculee(S1,Esp1,fct:agreg(Ag,V1,P1,V2,P2,V3,P3), fct:agreg("moy", Fiab1, NbTrait1, Fiab2, NbTrait2, Fiab3, NbTrait3), GrowCond1).**

R_2 : **aPourValeurServiceCalculee(S, Esp, V, Fiab, GrowCond)**
not aPourValeurServiceUtilisateur(S, Esp, V_u, F_u, GrowCond)
→ aPourValeurService(S, Esp, V, Fiab, GrowCond).

Pour composer les deux règles précédentes, on va unifier les atomes en vert avec l'unificateur suivant :

u = {(S,S1), (Esp,Esp1), (V,fct:agreg(Ag,V1,P1,V2,P2,V3,P3)),
(Fiab, fct:agreg("moy", Fiab1, NbTrait1, Fiab2, NbTrait2, Fiab3, NbTrait3)), (GrowCond, GrowCond1)}

On construit donc

$R_2 \circ R_1$: estLie3AFonctions(S1,F1,P1,F2,P2,F3,P3,Ag),
aPourValeurFonction(F1,Esp1,V1,Fiab1,GrowCond1),
aPourValeurFonction(F2,Esp1,V2,Fiab2,GrowCond1),
aPourValeurFonction(F3,Esp1,V3,Fiab3,GrowCond1),
nbrTrait(F1,NbTrait1), nbrTrait(F2,NbTrait2),
nbrTrait(F3,NbTrait3) not aPourValeurServiceUtilisateur(S1, Esp1, V_u, F_u, GrowCond1)

```
→ aPourValeurService(S1, Esp1, fct:agreg(Ag,V1,P1,V2,P2,V3,P3),
fct:agreg("moy",Fiab1,NbTrait1,Fiab2,NbTrait2,Fiab3,NbTrait3),
GrowCond1).
```

Les règles d'origine comportent en outre des prédicats calculés ($\neq$, $<$). Dans notre cas d'étude, nous choisissons d'ignorer les prédicats calculés dans l'explication car ils ne semblent pas pertinents (nous verrons un exemple à la fin du chapitre). Cependant, pour d'autres cas d'applications, nous pourrions considérer que tous les prédicats calculés sont des prédicats pertinents.

6.3.2 Gestion de la négation

Notre mécanisme d'explication permet de justifier la présence d'un fait, mais pas l'absence d'un fait. En effet, prouver qu'un fait n'est pas présent revient à vérifier qu'il n'existe aucune dérivation produisant ce fait, et donc énumérer un nombre exponentiel de dérivations. En plus, ces dérivations feraient intervenir aussi des atomes niés, il faudrait donc commencer par définir la forme que prendrait de telles explications.

Dans notre cas, pour gérer les atomes niés qui apparaissent dans les règles, on peut envisager plusieurs solutions : (1) soit dans les règles, les atomes niés ne portent que sur des prédicats pertinents, auquel cas le problème est résolu puisqu'on ne va pas chercher à remplacer ces atomes. Mais c'est très restrictif — et trop pour notre cas d'étude ; (2) soit on considère que ces atomes niés sont pertinents même s'ils ne portent pas sur des prédicats pertinents, mais le risque est de garder ou voir apparaitre des prédicats qui n'ont pas de sens pour l'utilisateur ; (3) soit ils sont ignorés dans les explications, l'inconvénient étant qu'on ne donne pas une explication complète à l'utilisateur.

Ici, nous avons choisi la dernière solution pour gérer la négation. En effet, notre objectif est de donner à l'utilisateur une information compréhensible même si ce n'est qu'une approximation du raisonnement effectué. Cela revient donc à ne considérer que le corps positif des règles. Par exemple, si on considère la règle $R : p(X,Y), not\ q(X) \rightarrow r(X,Y)$ avec $F = \{p(a,b)\}$, pour expliquer le fait $r(a,b)$, on donnera l'explication $(A,(R',\pi'))$ avec $A = \{p(a,b)\}$, $R' : p(X,Y) \rightarrow r(X,Y)$ et $\pi' = \{(X,a),(Y,b)\}$.

Exemple 6.3.3 *Dans l'exemple 6.3.2 précédent, si on veut donner une explication d'un atome α ayant pour prédicat* aPourValeurService *produite par la composition des règles R_1 et R_2, la partie en gris sera supprimée, et l'atome sera expliqué par un certain trigger ayant pour règle R' tel que corps(R') correspond au corps positif de la composée de $R_2 \circ R_1$*

R' : estLie3AFonctions(S1,F1,P1,F2,P2,F3,P3,Ag),
aPourValeurFonction(F1,Esp1,V1,Fiab1,GrowCond1),
aPourValeurFonction(F2,Esp1,V2,Fiab2,GrowCond1),
aPourValeurFonction(F3,Esp1,V3,Fiab3,GrowCond1),
nbrTrait(F1,NbTrait1), nbrTrait(F2,NbTrait2),
nbrTrait(F3,NbTrait3)
$\rightarrow$ aPourValeurService(S1, Esp1, fct:agreg(Ag,V1,P1,V2,P2,V3,P3),
fct:agreg("moy",Fiab1,NbTrait1,Fiab2,NbTrait2,Fiab3,NbTrait3),
GrowCond1).

Nous terminons ce chapitre par un exemple plus complet relatif à notre cas d'étude. On cherche maintenant à fournir une explication pertinente du fait α = aPourValeurService("nitrogen supply to the vine", "dactylis glomerata", 0.27, 78, "n").

On considère l'ensemble des prédicats pertinents suivants :

$\mathcal{V}$ = {aPourValeurService, aPourValeurFonction, aPourValeurTrait, estPositivementLieAFonction, estNegativementLieAFonction, estPositivementLieAService, estNegativementLieAService, aPourMethodeAgregation, nbrTrait}.

Nous donnons en figure 6.4, une représentation partielle du graphe de dépendance des ancêtres de α. Dans ce graphe, les ancêtres pertinents sont représentés en rouge. Dans la suite, nous allons dérouler le mécanisme de construction de l'explication pertinente du fait en vert, qui va correspondre à remonter aux ancêtres pertinents les plus proches (ce qui correspond à tous les ancêtres dans ce cas) et à construire la règle composée expliquant comment ces ancêtres pertinents mènent au fait à expliquer.

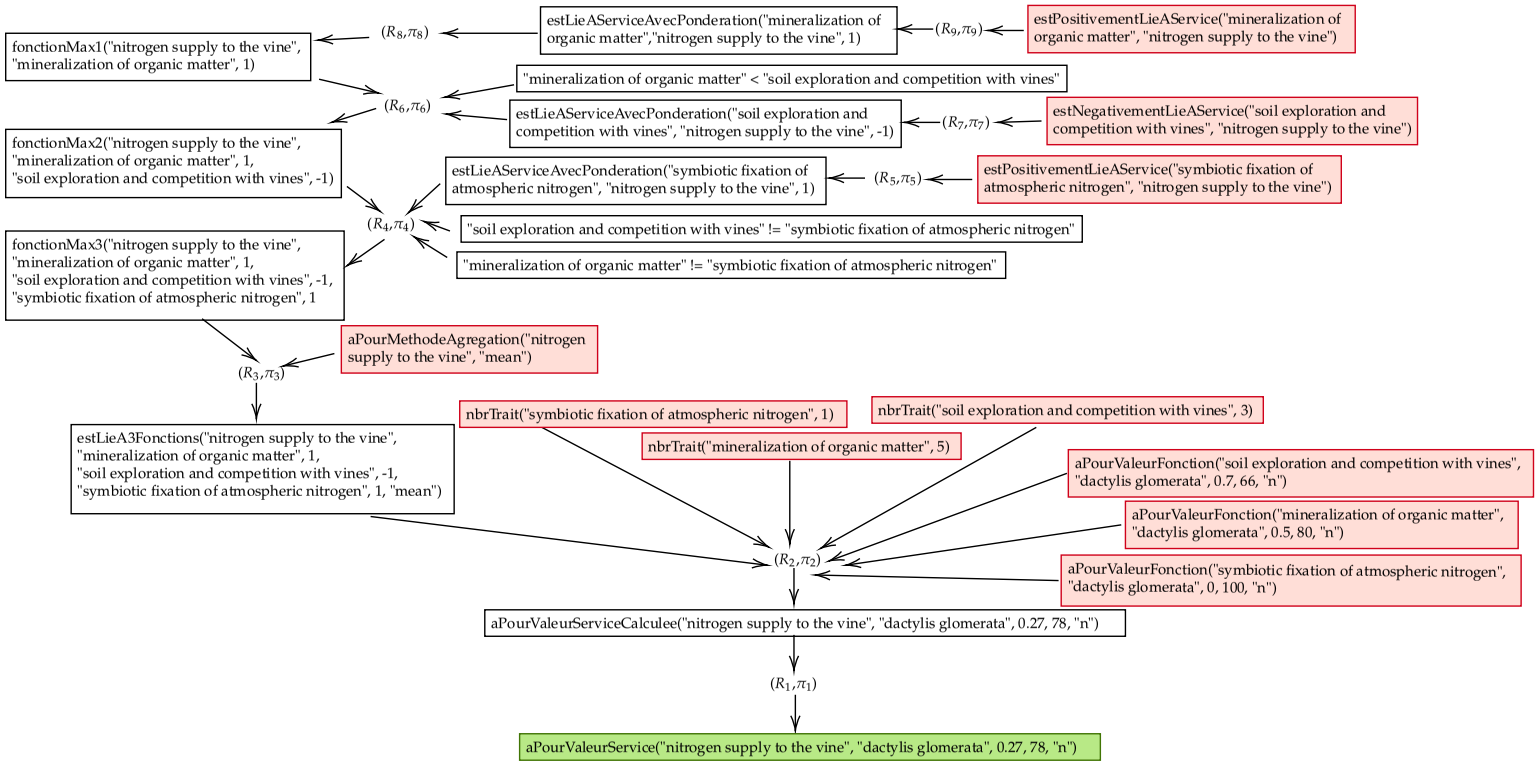


FIGURE 6.4 – Graphe (partiel) des ancêtres de l'atome `aPourValeurService("nitrogen supply to the vine", "dactylis glomerata", 0.27, 78, "n")`. En vert l'atome à expliquer, en rouge les atomes pertinents.

Le fait α est produit par le trigger (R_1, π_1) :

R_1 : **aPourValeurServiceCalculee(S, Esp, V, Fiab, GrowCond)**
not aPourValeurServiceUtilisateur(S, Esp, V_u, F_u, GrowCond)
$\rightarrow$ aPourValeurService(S, Esp, V, Fiab, GrowCond).

$\pi_1 = \{$(S,nitrogen supply to the vine), (Esp, dactylis glomerata), (V, 0.27), (Fiab, 78), (GrowCond, n)$\}$

Remarquons que corps(R_1) contient un atome (positif), mais qui n'est pas pertinent, et un atome nié, qui sera donc ignoré dans l'explication. Dans le but de trouver une explication pertinente de α, nous cherchons une explication du fait sur lequel s'envoie l'atome (positif) de corps(R_1), c'est-à-dire :

aPourValeurServiceCalculee("nitrogen supply to the vine", "dactylis glomerata", 0.27, 78, "n").

Ce fait est produit par le trigger (R_2, π_2) ci-dessous :

R_2:estLie3AFonctions(S1,F1,P1,F2,P2,F3,P3,Ag),
aPourValeurFonction(F1,Esp1,V1,Fiab1,GrowCond1),
aPourValeurFonction(F2,Esp1,V2,Fiab2,GrowCond1),
aPourValeurFonction(F3,Esp1,V3,Fiab3,GrowCond1),
nbrTrait(F1,NbTrait1), nbrTrait(F2,NbTrait2),
nbrTrait(F3,NbTrait3)
$\rightarrow$ **aPourValeurServiceCalculee(S1,Esp1,fct:agreg(Ag, V1, P1, V2, P2, V3, P3)**, fct:agreg("moy", Fiab1, NbTrait1, Fiab2, NbTrait2, Fiab3, NbTrait3),GrowCond1).

$\pi_2 = \{$(S1,nitrogen supply to the vine), (Esp1, dactylis glomerata), (Ag, mean), (GrowCond1, n), (F1, mineralization of organic matter), (P1, 1), (F2, soil exploration and competition with vines), (P2, -1), (F3, symbiotic fixation of atmospheric nitrogen), (P3, 1), (V1, 0.5), (V2, 0.7), (V3, 0), (Fiab1, 80), (Fiab2, 66), (Fiab3, 100), (NbTrait1, 5), (NbTrait2, 3), (NbTrait3, 1)$\}$

Le calcul de $R_1 \circ R_2 = R'$ a été expliqué dans l'exemple précédent. On obtient une nouvelle explication de α avec le trigger (R', π') :

R' : estLie3AFonctions(S1,F1,P1,F2,P2,F3,P3,Ag),
aPourValeurFonction(F1,Esp1,V1,Fiab1,GrowCond1),
aPourValeurFonction(F2,Esp1,V2,Fiab2,GrowCond1),
aPourValeurFonction(F3,Esp1,V3,Fiab3,GrowCond1),
nbrTrait(F3,NbTrait3), nbrTrait(F2,NbTrait2),
nbrTrait(F3,NbTrait3)
$\rightarrow$ aPourValeurService(S1, Esp1, fct:agreg(Ag,V1,P1,V2,P2,V3,P3),
fct:agreg(Ag,Fiab1,NbTrait1,Fiab2,NbTrait2,Fiab3,NbTrait3),
GrowCond1).

$\pi' = \pi_1 \circ \pi_2 = \{$(S1,nitrogen supply to the vine), (Esp1, dactylis glomerata), (Ag, mean), (GrowCond1, n), (F1, mineralization of organic matter), (P1, 1), (F2, soil exploration and competition with vines), (P2, -1), (F3, symbiotic fixation of atmospheric nitrogen), (P3, 1), (V1, 0.5), (V2, 0.7), (V3, 0) (Fiab1, 80), (Fiab2, 66), (Fiab3, 100), (NbTrait1,5), (NbTrait2,3), (NbTrait3,1)$\}$

Notons que l'atome nié est ignoré dans la composition. Et, parmi les atomes du corps de R', un seul n'est pas pertinent :

```
estLieA3Fonctions("nitrogen supply to the vine", "mineralization
of organic matter", 1, "soil exploration and competition with
vines", -1, "symbiotic fixation of atmospheric nitrogen", 1,
"mean")
```

Ce dernier est produit par le trigger (R_3, π_3) ci-dessous :

R_3 : fonctionMax3(S, F31, P31, F32, P32, F33, P33),
not fonctionMax4(S, F31, P31, F32, P32, F33, P33, X, Y),
aPourMethodeAgregation(S,Ag)
$\rightarrow$ estLieA3Fonctions(S, F31, P31, F32, P32, F33, P33, Ag)

π_3 = {(S,nitrogen supply to the vine), (F31, mineralization of organic matter), (P31, 1), (F32, soil exploration and competition with vines), (P32, -1), (F33, symbiotic fixation of atmospheric nitrogen), (P33, 1)}

Nous composons donc les règles R' et R_3 avec l'unificateur

u_3 = {(S,S1), (F31,F1), (P31,P1), (F32,F2), (P32,P2), (F31,F1), (P33,P3), (Ag, Ag)}

Nous composons ces règles et ne gardons que la partie positive, on obtient donc :

$R' \circ_{u_3} R_3$: fonctionMax3(S1, F1, P1, F2, P2, F3, P3),
aPourMethodeAgregation(S1,Ag), nbrTrait(F1,NbTrait1),
aPourValeurFonction(F1,Esp1,V1,Fiab1,GrowCond1),
aPourValeurFonction(F2,Esp1,V2,Fiab2,GrowCond1),
aPourValeurFonction(F3,Esp1,V3,Fiab3,GrowCond1),
nbrTrait(F2,NbTrait2), nbrTrait(F3,NbTrait3)
$\rightarrow$ aPourValeurService(S1, Esp1, fct:agreg(Ag,V1,P1,V2,P2,V3,P3),
fct:agreg(Ag,Fiab1,NbTrait1,Fiab2,NbTrait2,Fiab3,NbTrait3),
GrowCond1).

$\pi' \circ_{u_3} \pi_3$ = {(S1,nitrogen supply to the vine), (Esp1, dactylis glomerata), (Ag, mean), (GrowCond1, n), (F1, mineralization of organic matter), (P1, 1), (F2, soil exploration and competition with vines), (P2, -1), (F3, symbiotic fixation of atmospheric nitrogen), (P3, 1), (V1, 0.5), (V2, 0.7), (V3, 0) (Fiab1, 80),

```
(Fiab2, 66), (Fiab3, 100) (NbTrait1,5), (NbTrait2,3),
(NbTrait3,1)}
```

Nous continuons ces compositions tant qu'il existe un atome non pertinent dans le corps de la règle produite.On obtient finalement l'explication (A, T) visualisée 6.5 (qui correspond au graphe des ancêtres pertinents) tel que :

```
A = {aPourValeurFonction("mineralization of organic matter",
"dactylis glomerata", 0.5, 80, "n").
aPourValeurFonction("soil exploration and competition with vines",
"dactylis glomerata", 0.7, 66, "n").
aPourValeurFonction("symbiotic fixation of atmospheric nitrogen",
"dactylis glomerata", 0, 100, "n").
estPositivementLieAService("symbiotic fixation of atmospheric
nitrogen", "nitrogen supply to the vine").
estNegativementLieAService("soil exploration and competition
with vines", "nitrogen supply to the vine").
estPositivementLieAService("mineralization of organic matter",
"nitrogen supply to the vine").
aPourMethodeAgregation("nitrogen supply to the vine", "mean").
nbrTrait("mineralization of organic matter",5).
nbrTrait("soil exploration and competition with vines",3).
nbrTrait("symbiotic fixation of atmospheric nitrogen",1).}
```

et $T = (R, \pi)$ avec R qui est obtenue en prenant le corps positif et la tête de la règle composée

$$(R' \circ_{u_3} R_3) \circ_{u_4} R_4) \circ_{u_5} \ldots \circ_{u_9} R_9)$$

et

$$\pi = (\pi' \circ_{u_3} \pi_3) \circ_{u_4} \pi_4) \circ_{u_5} \ldots \circ_{u_9} \pi_9)$$

Si nous voulions considérer les prédicats calculés pertinents, nous ajouterons les faits suivants à l'ensemble A

```
"soil exploration and competition with vines" != "symbiotic
fixation of atmospheric nitrogen
mineralization of organic matter" != "symbiotic fixation of
atmospheric nitrogen"
"mineralization of organic matter" < "soil exploration and
competition with vines"
```

Mais comme nous pouvons le constater, ce sont des informations peu intéressantes pour un utilisateur.

Nous remarquons ici que le graphe des ancêtres pertinents (figure 6.5) encode des informations similaires à une partie d'un diagramme TFS appliqué à l'espèce

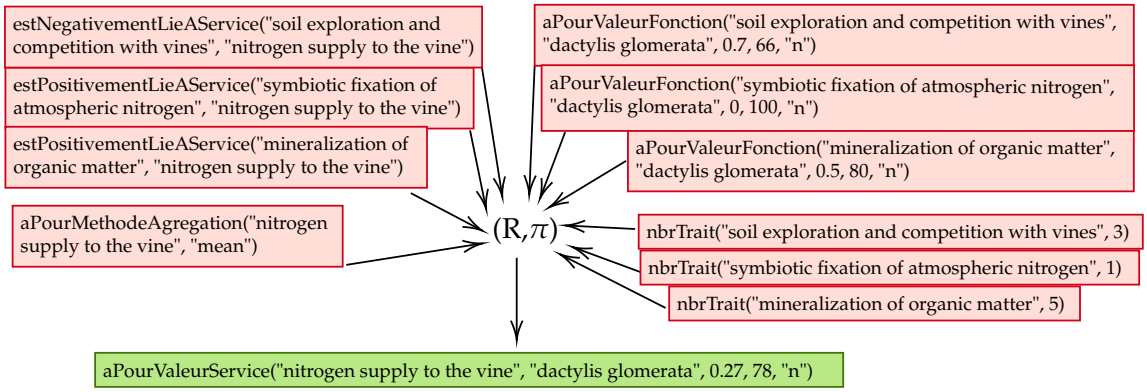


FIGURE 6.5 – Graphe des (plus proches) ancêtres pertinents du fait `aPourValeurService("nitrogen supply to the vine", "dactylis glomerata", 0.27, 78, "n")`, qui correspond à l'explication pertinente de ce fait.

en question. En effet, d'après le graphe, justifier la valeur d'un service pour une espèce, revient à donner les valeurs des fonctions qui lui sont liées. De même, si nous avions ensuite demandé des explications concernant les valeurs de fonctions nous aurions eu les valeurs des traits reliés. Finalement, fournir une explication des valeurs de fonctions et de services d'une espèce pourrait s'apparenter à fournir à l'utilisateur le diagramme TFS en ajoutant les valeurs des éléments du diagramme pour l'espèce considérée (voir figure 6.6). Ceci suggère une visualisation graphique naturelle des explications relatives au classement des espèces dans notre cas d'étude. De façon plus générale, le graphe des ancêtres pertinents paraît être un bon outil sur lequel s'appuyer pour visualiser les explications pertinentes.

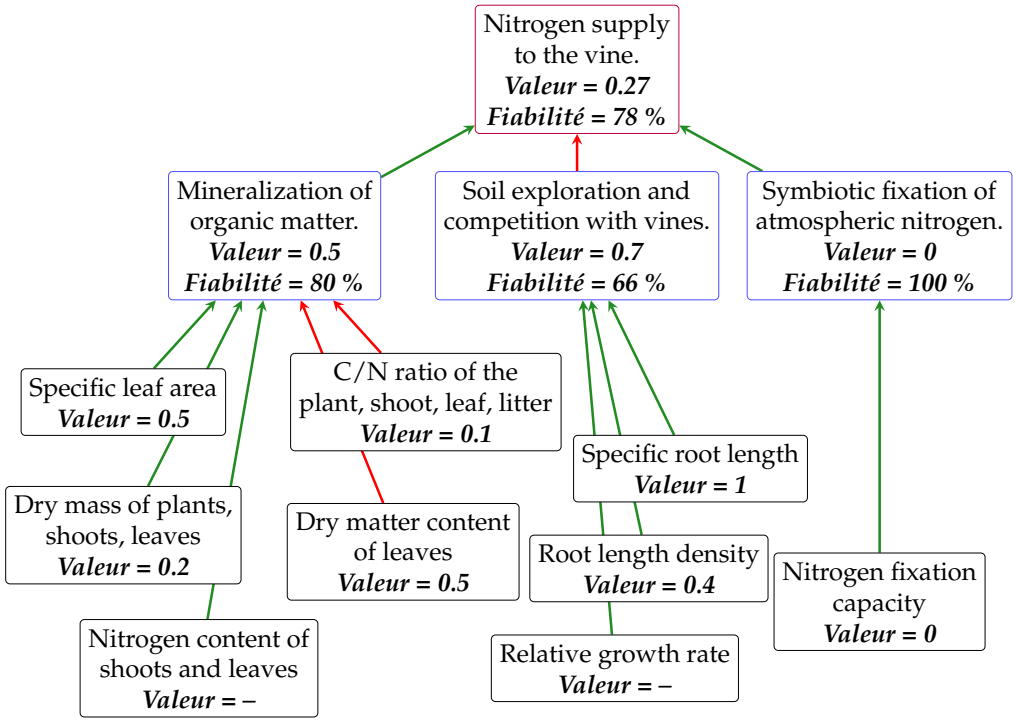


FIGURE 6.6 – Explication à l'utilisateur du fait `aPourValeurService("nitrogen supply to the vine", "dactylis glomerata", 0.27, 78, "n")` sous forme de diagramme TFS. Les valeurs sous les noms des traits, des fonctions et du service correspondent à celles de l'espèce *dactylis glomerata* en condition de croissance naturelle. Notons que "–" désigne une valeur manquante. Pour les fonctions et le service, la fiabilité de la valeur est mentionnée.

CHAPITRE 7

Conclusion et perspectives

Durant cette thèse, nous avons étudié la faisabilité d'une approche originale pour la *sélection de plantes de service* en agriculture basée sur les traits fonctionnels de ces plantes. L'objectif était *d'aider les agronomes à sélectionner des plantes à intégrer dans un système de culture* pour y rendre potentiellement des services écosystémiques souhaités, sans liste a priori des espèces à considérer.

Approche considérée L'idée principale était de combiner une représentation formelle des connaissances scientifiques en agronomie sur les relations entre traits fonctionnels, fonctions et services écosystémiques (que nous avons d'abord encodée sous forme de diagrammes) avec des sources de données construites indépendamment de ces diagrammes et fournissant des valeurs de traits fonctionnels pour un grand nombre d'espèces. Nous nous sommes en particulier appuyés sur la base de données TRY qui intègre les principaux ensembles de données sur les traits fonctionnels collectés par la communauté de recherche en écologie.

La mise en oeuvre de cette approche repose sur les principes du paradigme OBDA qui permet d'interroger des données à un niveau conceptuel, tout en tenant compte du raisonnement. Il s'agit, à notre connaissance, d'une nouvelle approche en matière d'ingénierie agroécologique. Par ailleurs, nous nous distinguons des systèmes OBDA proposés dans la littérature par une architecture s'appuyant sur différents types de mappings et un langage de représentation de connaissances plus expressif (Datalog). Nous avons décidé de matérialiser la base de connaissances, d'une part parce que c'est une technique qui est apparue adap-

tée au contexte de l'étude, d'autre part parce que l'utilisation de techniques de réécritures de requête aurait nécessité d'autres travaux théoriques en amont.

Propriétés de l'outil conçu Revenons aux sept exigences vis-à-vis de l'outil à concevoir que nous avions formulées dans l'introduction. Les quatre premiers points sont intrinsèquement liés à l'approche OBDA, à savoir :

1. *l'intégration de données*, qui peuvent provenir de sources hétérogènes et indépendantes,
2. la *combinaison des données* avec une représentation formelle des *connaissances expertes* qui permet le raisonnement,
3. la prise en compte des *raisonnements* lors du calcul des réponses à une requête,
4. la possibilité pour un utilisateur final de *formuler des requêtes* dans un vocabulaire familier.

Il nous reste à discuter des trois autres points : la *généricité* (ou généralité) de la formalisation, l' *évolutivité* du système et la capacité à *expliquer* à l'utilisateur des faits présents dans la base de faits saturée.

En ce qui concerne la généricité, les connaissances expertes sur les traits, les fonctions et les services sont formalisées sous la forme de deux types d'objets :

- Des *faits* qui décrivent des diagrammes spécifiques, y compris leurs liens avec les ID des bases de données.
- Des *règles* gérant le passage des valeurs de la base de données aux valeurs des traits, fonctions et services. Il est important de noter que ces règles sont génériques dans le sens où elles ne prennent pas en compte des traits, des fonctions ou des services spécifiques, ni des méthodes d'agrégation spécifiques. Elles ne sont donc pas liées à des diagrammes spécifiques, ni à des sources de données spécifiques.

Alors que généralement les systèmes basés sur des règles s'appuient sur des règles adaptées à un cas d'utilisation spécifique, nos règles pourraient en principe être applicables à n'importe quel cas d'utilisation qui suit le cadre trait-fonction-service.

Concernant l'évolutivité du système, la formalisation proposée permet une évolution des connaissances expertes encodées dans les diagrammes, sans impacter les règles. En effet, les changements sur les diagrammes, y compris l'introduction de nouveaux diagrammes, ne conduisent qu'à la construction d'une nouvelle base de faits initiale (en mettant à jour des faits experts). De même, les mises à

jour du contenu d'un fichier source ou d'une base de données ne conduisent qu'à déclencher à nouveau les mappings associés. Par ailleurs, l'introduction d'une nouvelle source de données nécessite (1) d'ajouter des mappings (éventuels) de nettoyage et des mappings vers la base de données de travail (working database), indépendamment des mappings déjà associés aux autres sources, et (2) de lier les traits fonctionnels des diagrammes experts à cette nouvelle source de données en identifiant les IDs de traits pertinents et en établissant la priorité de la source de données par rapports aux autres sources. Evidemment, la saturation de la base de faits doit être recalculée lorsque la base de faits initiale est changée.

Enfin, la dernière exigence consiste à être capable de fournir à l'utilisateur des explications relatives à la présence de faits dans la base de faits saturée. Comme nous l'avons vu, nous pouvions envisager deux façons extrêmes de fournir des explications : soit nous donnons les faits provenant de la base de faits initiale pour expliquer un fait déduit mais ce n'est pas très informatif puisqu'on ne sait rien des règles qui interviennent dans la dérivation ; soit nous donnons des explications directes en remontant étape par étape la dérivation (plus précisément, le graphe de dépendance des atomes), mais ce processus peut devenir très long et n'a pas forcément de sens pour l'utilisateur. Cherchant un compromis entre ces deux extrêmes, nous avons abouti à la notion d'explication pertinente, qui requiert d'identifier la partie du vocabulaire qui a du sens pour l'utilisateur : une explication pertinente n'utilise que ce vocabulaire, ce qui nécessite de composer des applications de règles.

Difficultés rencontrées Pour mettre notre approche en pratique, nous avons d'abord dû déployer d'importants efforts pour nettoyer les données de TRY (reçues en format csv). Il s'agissait d'une étape obligatoire avant toute exploitation automatisée des données.Par la suite, nous avons été confrontés au problème majeur des valeurs manquantes. Ce problème, analysé en détails dans [Kat+20], est peut-être exacerbé par le fait que les données ont été collectées pour des recherches en écologie, qui se concentrent davantage sur les espèces naturelles et spontanées que sur les espèces cultivées à des fins agricoles.

Pour atténuer l'impact des valeurs manquantes, nous avons exploité les IDs de traits interchangeables et nous avons introduit un paramètre de fiabilité, qui permet de considérer aussi les espèces pour lesquelles une partie des traits pertinents n'ont pas de valeurs. L'évaluation réalisée montre que des résultats très satisfaisants peuvent être obtenus tant que la proportion de valeurs manquantes n'est pas trop élevée. Puisque l'initiative TRY se développe, tant en volume qu'en

standardisation des données, et qu'il y a un effort croissant pour produire des données sur les traits fonctionnels en agronomie [Ble18 ; Gar+20 ; Gar+19 ; Woo+15 ; MI15 ; MI18], on ne peut que s'attendre à des résultats plus précis dans les prochaines années.

En dehors des valeurs manquantes, une raison des divergences observées entre les résultats de l'outil et les résultats de la littérature semble être, pour certains traits, une forte variation des valeurs mesurées pour des plantes d'une même espèce. Outre les problèmes potentiels de fiabilité de certaines observations, il est probable que ces variations soient liées à des différences dans les contextes des observations, qu'il s'agisse du contexte de culture (comme les caractéristiques du sol, le climat et la conduite technique, ...) ou du stade de croissance de la plante. En dehors des conditions de croissance des plantes, nous n'avons pas pris en compte le contexte des observations pour deux raisons : premièrement, cela aurait réduit le nombre d'observations pertinentes, et donc aggravé le problème des valeurs manquantes ; deuxièmement, de nombreuses observations dans TRY ne sont pas fournies avec des informations contextuelles adéquates (sans parler du problème de l'exploitation automatique de ces informations). Une fois encore, le développement des données sur les traits devrait permettre d'exploiter les informations contextuelles dans un avenir proche.

Perspectives Nous listons maintenant quelques unes des perspectives qui nous paraissent les plus intéressantes à explorer. Nous commençons par des extensions de notre modélisation (directement liées au cas d'étude) puis nous présenterons des perspectives plus générales en informatique.

Une première perspective d'extension de notre modélisation serait de prendre en compte les *informations contextuelles*. On pourrait prendre en compte 2 types d'informations contextuelles :

(i) Les *informations contextuelles liées aux conditions de la parcelle*, par exemple : le climat, les caractéristiques du sol, etc. Dans ce cas, notre objectif serait de restreindre le contexte des observations déposées dans les bases de données au contexte de la parcelle (dans laquelle les plantes de service doivent être implantées) afin qu'ils soient les plus *proches* possible. Une façon de mettre en place cet aspect serait, pour chaque information contextuelle :

- regrouper les valeurs possibles. Par exemple, concernant le climat, on considèrerait différentes valeurs possibles : méditerranéen, tropical, désertique, etc ;

- préciser le contexte de la parcelle. Ce sera une entrée possible de l'utilisateur qui génèrera des connaissances sous forme de faits. Par exemple, si l'utilisateur est dans un contexte méditerranéen, nous construirons le fait `climatParcelle(mediterraneen);`
- récupérer pour chaque observation le contexte dans lequel elle se situe. Ceci peut se faire directement si l'information est présente, sinon, peut être déduit. Par exemple, grâce à la zone géographique, nous pourrons déduire le type de climat et le fait `climatObservation(Observation, Climat)` sera produit;
- enfin, on se restreint aux observations ayant un contexte semblable à celui de la parcelle.

(*ii*) Les *informations contextuelles liées à la réponse des plantes à l'environnement* (à laquelle correspond des traits de réponse) [DNG18], par exemple : la résistance au gel, ou à la sécheresse, ou la capacité à pousser dans un sol pauvre en nutriments, etc. Ce type de contexte est indépendant du contexte de la parcelle. Les traits de réponse (comme la résistance au gel ou à la sécheresse, ou la capacité à pousser dans un sol pauvre en nutriments) peuvent être pris en compte a posteriori pour filtrer les espèces retournées comme réponses. Mais également avant le raisonnement, afin de se restreindre dès le départ aux espèces ayant les propriétés désirées.

Notre cadre permet de telles extensions, le facteur limitant étant la *disponibilité des données*. De même, notre modélisation laisse la possibilité de raffiner les diagrammes d'experts, notamment en pondérant les contributions des traits aux fonctions, et des fonctions aux services et en rendant les méthodes d'agrégation plus complexes, le facteur limitant étant de savoir si les connaissances de pointe du domaine permettent de tels raffinements.

Une autre perspective d'extension serait d'intégrer dans le raisonnement les *pratiques agricoles*. Les pratiques agricoles sont l'ensemble des techniques utilisées par les cultivateurs lors de l'exploitation d'une parcelle pour gérer la production agricole et son environnement. On pourrait supposer qu'une fonction potentiellement remplie par une espèce peut être inhibée par certaines pratiques agricoles. Par exemple, une espèce peut être potentiellement en compétition avec la vigne pour une ressource (nutriments, eau), sauf si elle n'est pas présente quand la vigne a un grand besoin de la ressource. On pourrait donc affirmer que cette espèce rend généralement bien un service écosystémique pour la vigne, sauf dans une période de compétition pendant laquelle elle doit être retirée (par exemple, par une pratique agricole qui la détruit). On voit que la prise en compte des pra-

tiques agricoles comporte des aspects temporels liés au cycle de vie des plantes et au climat (notamment, pour savoir à quel moment il y a compétition entre des espèces). Nous pourrions prendre en compte les pratiques agricoles dans le raisonnement comme étant des éléments qui ont un impact (positif ou négatif) sur les fonctions ou même les services. Dans ce cas, un impact positif pourra être considéré comme un bonus sur les valeurs de fonctions ou services et, a contrario, un impact négatif, pourra être considéré comme un malus sur les valeurs de fonctions ou services.

Par ailleurs, dans la représentation actuelle, les espèces sont considérées indépendamment les unes des autres. Il serait intéressant de proposer un *bouquet d'espèces* qui pourraient se compléter pour remplir des services écosystémiques. Supposons par exemple que l'on souhaite sélectionner des espèces pour le rendu des 2 services suivants : (1) l'amélioration de la structure du sol et (2) la fixation et le recyclage de l'azote. Grâce à leurs racines pivotantes et fortes, les crucifères (radis, moutarde, etc.) rendent le premier. Grâce à leur capacité de fixation de l'azote, les légumineuses rendent le second. Il faudrait donc pouvoir associer des espèces de ces 2 familles (il est aussi possible que certaines espèces rendent les 2 services ; par exemple, la luzerne a un système racinaire vigoureux et profond et c'est une légumineuse). Une difficulté sur laquelle il faudrait travailler est donc celle du choix des bonnes combinaisons d'espèces. En effet, si l'outil identifie les espèces A, B et C pour le service 1 et B, D et E pour le service 2, on peut soit retenir l'espèce B seule ; soit mélanger $A + D$ ou $A + E$ ou $C + D$ ou $C + E$. A priori, pour que le mélange de 2 espèces fonctionne, il faudrait qu'elles répondent de la même manière aux conditions d'environnement (sol, climat) et de conduite technique. Par exemple, qu'elles soient également tolérantes à la sécheresse, sinon c'est la plus tolérante qui dominera. Il faudrait donc qu'elles aient des traits de réponse semblables. On pourrait aussi souhaiter qu'il y ait une complémentarité entre les 2 espèces, par exemple que leurs systèmes racinaires n'explorent pas les mêmes horizons de sol. En pratique, cela pourrait se traduire par des contraintes à imposer sur certains traits fonctionnels des espèces à associer empêchant ainsi un écart trop important quant à leur réponse aux conditions d'environnement ou au contraire une trop grande similarité dans leur capacité d'accès aux ressources. D'une manière générale, comme dans tout mélange d'espèces, il faut veiller à ce qu'il y ait "facilitation" (les 2 espèces associées poussent aussi bien voire mieux que quand elles sont isolées) et non "compétition" (les 2 espèces associées, ou l'une d'elles, poussent moins bien que quand elles sont isolées) [Mal+09].

Pour mettre en oeuvre cela, il faut qu'il y ait suffisamment de données et

de travaux sur la compatibilité entre les espèces. Même si nous pouvons trouver des données expérimentales sur les performances (rendement, sensibilité aux maladies, etc) des mélanges d'espèces (herbacées ou ligneuses, par exemple en agroforesterie), éventuellement des simulations à partir de modèles de culture [Gau+19], ces données restent encore insuffisantes. De même, il y a des études qui traitent des mélanges d'espèces (ou de variétés) qui sont devenus un sujet important en agroécologie. Mais, il n'y a pour l'instant que très peu de travaux sur la conception de mélanges.

Du point de vue de la discipline informatique, le cas d'étude en agroécologie nous a conduit à étendre les cadres méthodologiques et théoriques existants, en affinant le cadre OBDA avec différents types de mappings dotés d'une forme commune, et en proposant une notion d'explication pertinente, ce qui ouvre également des perspectives. En ce qui concerne les mappings, on pourrait effectuer un état de l'art des mappings utilisés en pratique dans le contexte d'accès aux données et définir une forme commune qui couvre les différentes catégories de mappings identifiées. Ceci permettrait la définition de traitements génériques sur les mappings, qui faciliterait l'intégration de différentes sources de données. En ce qui concerne les explications, nous avons introduit la notion d'explication pertinente pour expliquer un fait inféré dans un vocabulaire compréhensible par l'utilisateur. Il reste à implémenter et expérimenter le mécanisme défini, en particulier sur le cas d'étude, ce qui nécessite de mettre en place une interface graphique adéquate. En outre, il serait intéressant de répertorier les types de questions intéressant prioritairement un utilisateur de façon à fournir des explications sous une forme adaptée à la question. Par exemple, en ce qui concerne la question "pourquoi telle espèce a-t-elle telle valeur de service ?", nous avons proposé de nous appuyer sur le diagramme TFS instancié pour cette espèce pour présenter les explications à l'utilisateur ; d'autres questions pertinentes pourraient par exemple concerner la fiabilité des calculs ou les raisons pour lesquelles une espèce est jugée meilleure qu'une autre pour un certain service, auquel cas d'autres formes de restitution des explications pourraient être envisagées. Concernant le cadre théorique des explications, nous avons écarté les atomes niés et les prédicats calculés apparaissant dans les règles. Comment construire des explications plus précises en exploitant ces éléments reste une question ouverte. Cette question est également liée à la notion d'explication de l'absence d'un fait, que nous n'avons pas considérée.

D'autre part, nous avons suivi une approche de matérialisation de la base de faits. Or, il est clair que des mécanismes de réécriture de requête devraient égale-

ment être envisagés dans la perspective de l'intégration d'autres sources de données. Par exemple, imaginons que l'on veuille intégrer une base de données qui fournit le climat associé à un point géographique, de façon à déterminer le climat associé à une observation géolocalisée. Il n'est pas envisageable de matérialiser par avance le climat associé à tous les points imaginables. Etant donné un mapping associé à cette base de données et fournissant le climat correspondant à une localisation, on voudrait plutôt sélectivement interroger la base de données via ce mapping. Par ailleurs, la définition d'un contexte d'installation par l'utilisateur (par exemple concernant le climat) peut être considérée comme une requête à prendre en compte dès la construction de la base de travail (working database), de façon à se restreindre aux données d'intérêt. Ceci suggère également un mécanisme de réécriture de requête, de façon à traduire les critères de l'utilisateur exprimés dans le vocabulaire ontologique en des requêtes sur les bases de données. Plus généralement, deux problématiques restent largement à explorer : comment combiner la matérialisation de certaines sources de données avec la virtualisation d'autres sources, et comment prendre en compte au niveau des mappings de données des critères définis au niveau conceptuel. Ceci pose de multiples questions d'ordre méthodologique, théorique et algorithmique.

Annexes

ANNEXE A

TRY : les 52 traits demandés

TraitID	TraitName	% de taxons	% d'herbacés
1021	Plant carbon/nitrogen (C/N) ratio	0.01	100
409	Shoot carbon/nitrogen (C/N) ratio	1.36	60
146	Leaf carbon/nitrogen (C/N) ratio	3.12	39
150	Litter carbon/nitrogen (C/N) ratio	0.1	21
77	Plant growth rate relative (plant relative growth rate, RGR)	0.56	61
8	Plant nitrogen(N) fixation capacity	8.89	34
700	Plant biomass and allometry : Plant dry mass	0.15	83
388	Plant biomass and allometry : Leaf dry mass	0.04	21
403	Plant biomass and allometry : Shoot dry mass (plant aboveground dry mass) per plant	0.5	90
1508	Root length per soil volume	0.02	42
2025	Fine root length per soil volume	0.03	69
2281	Fine root (absorptive) length per soil volume	0.02	4
		Suite ⇒	

Table A.1 (Suite)

TraitID	TraitName	% de taxons	% d'herbacés
3086	Leaf area per leaf dry mass (specific leaf area, SLA or 1/LMA) petiole, rhachis and midrib excluded	0.27	22
3117	Leaf area per leaf dry mass (specific leaf area, SLA or 1/LMA) : undefined if petiole is in- or excluded	8.3	36
3116	Leaf area per leaf dry mass (specific leaf area, SLA or 1/LMA) : petiole included	4.4	53
3115	Leaf area per leaf dry mass (specific leaf area, SLA or 1/LMA) : petiole excluded	4.9	44
47	Leaf dry mass per leaf fresh mass (leaf dry matter content, LDMC)	4.3	59
1080	Root length per root dry mass (specific root length, SRL)	0.33	81
614	Fine root length per fine root dry mass (specific fine root length, SRL)	0.7	56
339	Shoot nitrogen (N) content per shoot dry mass	0.13	97
408	Plant biomass and allometry : Shoot nitrogen (N) content per plant	0.08	98
502	Plant biomass and allometry : Leaf nitrogen (N) content per plant	0.08	50
1126	Shoot organic nitrogen (N) content per shoot dry mass	0.08	80
82	Root tissue density (root dry mass per root volume)	0.26	92.98
116	Root dry mass per root volume (root density, root tissue density)	0.01	100
83	Root diameter	0.26	95.4
86	Root soluble components content per root dry mass	0.06	97.98
88	Root lignin content per root dry mass	0.09	89.85
119	Plant biomass and allometry : Plant dry mass per plant fresh mass	0.01	100
		Suite ⇒	

Table A.1 (Suite)

TraitID	TraitName	% de taxons	% d'herbacés
6	Root rooting depth	2.46	57.1
21	Stem diameter	2.18	10.44
410	Leaf area per plant	0.45	37.3
587	Plant growth rate	3.52	62.42
9	Root/shoot ratio	0.37	82.1
896	Fine root diameter	0.58	57.08
1430	Mycorrhizal colonization AM : fraction of root length that contains arbuscular mycorrhizal structures	0.14	54.93
1435	Mycorrhizal colonization : fraction of root length with mycorrhizla hyphae, arbuscules, or vesicles	0.12	60.22
1389	Root length relative growth rate	0.005	100
42	Plant growth form	97.51	47.56
343	Plant life form (Raunkiaer life form)	7.94	67.59
3440	Plant biomass and allometry : Crown (canopy) area : projected as seen from above	0.19	9.3
3459	Plant belowground relative growth rate	0.003	100
540	Leaf scandium (Sc) content per leaf dry mass	0.42	14.33
660	Leaf nitrogen (N) content organic per leaf dry mass	0.01	57.89
668	Plant nitrogen (N) content per plant dry mass	0.01	0
9	Root/shoot ratio	0.37	82.1
954	Litter (leaf) nitrogen (N) content per leaf litter dry mass	0.01	0
1410	Root growth rate : fine root length production per day per coarse root length	0.004	0
1429	Mycorrhizal colonization : fraction of root tips that contain mycorrhizal structures	0.08	2.5
1778	Fine root/shoot ratio	0.05	59.74
		Suite ⇒	

Table A.1 (Suite)

TraitID	TraitName	% de taxons	% d'herbacés
1930	Fine root length relative growth rate	0.0007	0
2186	Fine root (absorptive) length relative growth rate	0.003	0

TABLE A.1 – 52 traits demandés dans TRY avec leurs IDs. Pour chaque trait, nous avons le pourcentage de taxons (parmi les 150976) et le pourcentage d'herbacés (par rapport au pourcentage de taxons)

ANNEXE B

Base de connaissances : Règles

Ensemble de règles

Ci-dessous, les règles sont dans le format DLGP (Datalog Plus) de Graal. Une règle $C \rightarrow T$ s'écrit $T : -C$.

Règles hiérarchie de concepts

```
top(X):-objetTFS(X).
top(X):-espece(X).
top(X):-agregation(X).
top(X):-source(X).
top(X):-conditionDeCroissance(X).
objetTFS(X):-fonction(X).
objetTFS(X):-service(X).
objetTFS(X):-trait(X).
source(X):-sourceDeDonnee(X).
source(X):-sourceUtilisateur(X).
service(X):-serviceBiologique(X).
service(X):-serviceBioPhysiqueDuCadreDeVie(X).
service(X):-serviceDeLaProductionAgricole(X).
```

Règles hiérarchie de relations

```
estLieAFonctionAvecPonderation(X,Y,1):-estPositivementLieAFonction(X,Y).
estLieAFonctionAvecPonderation(X,Y,-1):-estNegativementLieAFonction(X,Y).
estLieAServiceAvecPonderation(X,Y,1):-estPositivementLieAService(X,Y).
estLieAServiceAvecPonderation(X,Y,-1):-estNegativementLieAService(X,Y).
estLieAFonction(X,Y):-estLieAFonctionAvecPonderation(X,Y,P).
estLieAService(X,Y):-estLieAServiceAvecPonderation(X,Y,P).
```

Règles signatures des relations

```
trait(X):-estLieAFonction(X,Y).
fonction(Y):-estLieAFonction(X,Y).
fonction(X):-estLieAService(X,Y).
service(Y):-estLieAService(X,Y).
objetTFS(X):-aPourMethodeAgregation(X,Y).
agregation(Y):-aPourMethodeAgregation(X,Y).
trait(X):-aPourTraitID(X,Y,Z).
sourceDeDonnee(Z):-aPourTraitID(X,Y,Z).
source(X):-basePrioritaire(X,Y).
source(Y):-basePrioritaire(X,Y).
source(X):-traitPrioritaireBase(X,Y).
source(Y):-traitPrioritaireBase(X,Y).
trait(X1):-aPourValeurTraitBase(X1,X2,X3,Valeur,X5).
espece(X2):-aPourValeurTraitBase(X1,X2,X3,Valeur,X5).
conditionDeCroissance(X3):-aPourValeurTraitBase(X1,X2,X3,Valeur,X5).
source(X5):-aPourValeurTraitBase(X1,X2,X3,Valeur,X5).
trait(X1):-aPourValeurTrait(X1,X2,X3,Valeur).
espece(X2):-aPourValeurTrait(X1,X2,X3,Valeur).
conditionDeCroissance(X3):-aPourValeurTrait(X1,X2,X3,Valeur).
fonction(X1):-aPourValeurFonction(X1,X2,Valeur,Fiabilite,X3).
espece(X2):-aPourValeurFonction(X1,X2,Valeur,Fiabilite,X3).
conditionDeCroissance(X3):-aPourValeurFonction(X1,X2,Valeur,Fiabilite,X3).
fonction(X1):-aPourValeurFonctionUtilisateur(X1,X2,Valeur,Fiabilite,X3).
espece(X2):-aPourValeurFonctionUtilisateur(X1,X2,Valeur,Fiabilite,X3).
conditionDeCroissance(X3):-aPourValeurFonctionUtilisateur(X1,X2,Valeur,
Fiabilite,X3).
```

```
fonction(X1):-aPourValeurFonctionCalculee(X1,X2,Valeur,Fiabilite,X3).
espece(X2):-aPourValeurFonctionCalculee(X1,X2,Valeur,Fiabilite,X3).
conditionDeCroissance(X3):-aPourValeurFonctionCalculee(X1,X2,Valeur,
Fiabilite,X3).
service(X1):-aPourValeurService(X1,X2,Valeur,Fiabilite,X3).
espece(X2):-aPourValeurService(X1,X2,Valeur,Fiabilite,X3).
conditionDeCroissance(X3):-aPourValeurService(X1,X2,Valeur,Fiabilite,X3).
service(X1):-aPourValeurServiceUtilisateur(X1,X2,Valeur,Fiabilite,X3).
espece(X2):-aPourValeurServiceUtilisateur(X1,X2,Valeur,Fiabilite,X3).
conditionDeCroissance(X3):-aPourValeurServiceUtilisateur(X1,X2,Valeur,
Fiabilite,X3).
service(X1):-aPourValeurServiceCalculee(X1,X2,Valeur,Fiabilite,X3).
espece(X2):-aPourValeurServiceCalculee(X1,X2,Valeur,Fiabilite,X3).
conditionDeCroissance(X3):-aPourValeurServiceCalculee(X1,X2,Valeur,
Fiabilite,X3).
```

Règles pour la construction des prédicats de haute arité

```
%%%% Règles pour la construction des faits estLieAkTraitIDs
traitIDRestant1(T, ID1, Source) :- aPourTraitID(T, ID1, Source),
aPourTraitID(T, ID2, Source), pred:inf(ID1,ID2).
traitIDMax1(T, ID1, Source) :- aPourTraitID(T, ID1, Source),
not traitIDRestant1(T, ID1, Source).

traitIDRestant2(T, ID1, Source) :- aPourTraitID(T, ID1, Source),
 aPourTraitID(T, ID2, Source), pred:inf(ID1,ID2),
 traitIDMax1(T,ID3, Source), not pred:isEqualToString(ID3,ID2).
traitIDMax2(T, ID1, ID2, Source) :- traitIDMax1(T, ID1, Source),
aPourTraitID(T, ID2, Source), pred:inf(ID2,ID1),
not traitIDRestant2(T,ID2,Source).


traitIDRestant3(T, ID1, Source) :- aPourTraitID(T, ID1, Source),
 aPourTraitID(T, ID2, Source), pred:inf(ID1,ID2),
 traitIDMax2(T, ID3, ID4, Source), not pred:isEqualToString(ID3, ID2),
 not pred:isEqualToString(ID4,ID2).
traitIDMax3(T, ID1, ID2, ID3, Source) :- traitIDMax2(T, ID1, ID2, Source),
aPourTraitID(T, ID3, Source),not pred:isEqualToString(ID3,ID1),
```

```
not pred:isEqualToString(ID3,ID2), not traitIDRestant3(T,ID3,Source).

traitIDRestant4(T, ID1, Source) :- aPourTraitID(T, ID1, Source),
aPourTraitID(T, ID2, Source), pred:inf(ID1,ID2),
traitIDMax3(T, ID3, ID4, ID5, Source),
not pred:isEqualToString(ID3,ID2),
not pred:isEqualToString(ID4,ID2),
not pred:isEqualToString(ID5,ID2).
traitIDMax4(T, ID1, ID2, ID3, ID4, Source)
:- traitIDMax3(T, ID1, ID2, ID3, Source),
aPourTraitID(T, ID4, Source),
not pred:isEqualToString(ID4,ID1),
not pred:isEqualToString(ID4,ID2),
not pred:isEqualToString(ID4,ID3),
not traitIDRestant4(T,ID4,Source).

traitIDRestant5(T, ID1, Source) :- aPourTraitID(T, ID1, Source),
aPourTraitID(T, ID2, Source),
pref:inf(ID1, ID2), traitIDMax4(T, ID3, ID4, ID5, ID6, Source),
not pred:isEqualToString(ID3,ID2),
 not pred:isEqualToString(ID4,ID2),
 not pred:isEqualToString(ID5,ID2),
 not pred:isEqualToString(ID6,ID2).
traitIDMax5(T, ID1, ID2, ID3, ID4, ID5, Source)
:- traitIDMax4(T, ID1, ID2, ID3, ID4, Source),
aPourTraitID(T, ID5, Source),
not pred:isEqualToString(ID5,ID1),
not pred:isEqualToString(ID5,ID2),
not pred:isEqualToString(ID5,ID3),
not pred:isEqualToString(ID5,ID4),
not traitIDRestant5(T,ID5,Source).


estLieA1TraitID(T,ID1,Source) :- traitIDMax1(T, ID1, Source),
not traitIDMax2(T, ID1, X, Source).

estLieA2TraitsID(T,ID1,ID2,Source,Ag) :- traitIDMax2(T, ID1, ID2, Source),
```

```
not traitIDMax3(T, ID1, ID2, X, Source),
aPourMethodeAgregation(T,Ag).

estLieA3TraitsID(T,ID1,ID2,ID3,Source,Ag) :-
traitIDMax3(T, ID1, ID2, ID3, Source),
not traitIDMax4(T, ID1, ID2,ID3, X, Source),
aPourMethodeAgregation(T,Ag).

estLieA4TraitsID(T,ID1,ID2,ID3,ID4,Source,Ag) :-
traitIDMax4(T, ID1, ID2, ID3, ID4, Source),
not traitIDMax5(T, ID1, ID2,ID3,ID4, X, Source),
aPourMethodeAgregation(T,Ag).

estLieA5TraitsID(T,ID1,ID2,ID3,ID4,ID5,Source,Ag) :-
traitIDMax5(T, ID1, ID2, ID3, ID4, ID5, Source),
not traitIDMax6(T, ID1, ID2, ID3, ID4, ID5, X, Source),
aPourMethodeAgregation(T,Ag).


%%%% Règles pour la construction des faits estLieAkTraits
traitsRestants1(F, T1, P1)
:- estLieAFonctionAvecPonderation(T1, F, P1),
 estLieAFonctionAvecPonderation(T2, F, P2),
 pred:inf(T1,T2).
traitMax1(F, T1, P1)
:- estLieAFonctionAvecPonderation(T1, F, P1),
not traitsRestants1(F, T1, P1).

traitsRestants2(F, T1, P1)
:- estLieAFonctionAvecPonderation(T1,F,P1),
estLieAFonctionAvecPonderation(T2,F,P2),
pred:inf(T1,T2), traitMax1(F,T3,P3),
not pred:isEqualToString(T3,T2).
traitMax2(F, T1, P1, T2, P2) :- traitMax1(F,T1,P1),
estLieAFonctionAvecPonderation(T2,F,P2),
pred:inf(T2,T1), not traitsRestants2(F,T2,P2).
```

```
traitsRestants3(F, T1, P1)
:- estLieAFonctionAvecPonderation(T1,F,P1),
estLieAFonctionAvecPonderation(T2,F,P2),
pred:inf(T1,T2), traitMax2(F,T3,P3,T4,P4),
not pred:isEqualToString(T3,T2),
not pred:isEqualToString(T4,T2).
traitMax3(F, T1, P1, T2, P2, T3, P3)
:- traitMax2(F,T1,P1,T2,P2),
estLieAFonctionAvecPonderation(T3,F,P3),
not pred:isEqualToString(T1,T3),
not pred:isEqualToString(T2,T3),
not traitsRestants3(F,T3,P3).

traitsRestants4(F, T1, P1)
:- estLieAFonctionAvecPonderation(T1,F,P1),
estLieAFonctionAvecPonderation(T2,F,P2),
pred:inf(T1,T2),
traitMax3(F,T3,P3,T4,P4,T5,P5),
not pred:isEqualToString(T3,T2),
not pred:isEqualToString(T4,T2),
not pred:isEqualToString(T5,T2).
traitMax4(F, T1, P1, T2, P2, T3, P3, T4, P4)
:- traitMax3(F,T1,P1,T2,P2,T3,P3),
estLieAFonctionAvecPonderation(T4,F,P4),
not pred:isEqualToString(T1,T4),
not pred:isEqualToString(T2,T4),
not pred:isEqualToString(T3,T4),
not traitsRestants4(F,T4,P4).

traitsRestants5(F, T1, P1)
:- estLieAFonctionAvecPonderation(T1,F,P1),
estLieAFonctionAvecPonderation(T2,F,P2),
pred:inf(T1,T2),
traitMax4(F,T3,P3,T4,P4,T5,P5,T6,P6),
not pred:isEqualToString(T3,T2),
not pred:isEqualToString(T4,T2),
not pred:isEqualToString(T5,T2),
```

```
not pred:isEqualToString(T6,T2).
traitMax5(F, T1, P1, T2, P2, T3, P3, T4, P4, T5, P5)
:- traitMax4(F,T1,P1,T2,P2,T3,P3,T4,P4),
estLieAFonctionAvecPonderation(T5,F,P5),
not pred:isEqualToString(T1,T5),
not pred:isEqualToString(T2,T5),
not pred:isEqualToString(T3,T5),
not pred:isEqualToString(T4,T5),
not traitsRestants5(F,T5,P5).

estLieA1Trait(F, T1, P1) :- traitMax1(F, T1, P1),
not traitMax2(F, T1, P1, X, Y).

estLieA2Traits(F, T1, P1, T2, P2, Ag)
:- traitMax2(F, T1, P1, T2, P2),
not traitMax3(F, T1, P1, T2, P2, X, Y),
aPourMethodeAgregation(F,Ag).

estLieA3Traits(F, T1, P1, T2, P2, T3, P3, Ag)
:- traitMax3(F, T1, P1, T2, P2, T3, P3),
not traitMax4(F, T1, P1, T2, P2, T3, P3, X, Y),
aPourMethodeAgregation(F,Ag).

estLieA4Traits(F, T1, P1, T2, P2, T3, P3, T4, P4, Ag)
:- traitMax4(F, T1, P1, T2, P2, T3, P3, T4, P4),
not traitMax5(F, T1, P1, T2, P2, T3, P3, T4, P4, X, Y),
aPourMethodeAgregation(F,Ag).

estLieA5Traits(F, T1, P1, T2, P2, T3, P3, T4, P4, T5, P5, Ag)
:- traitMax5(F, T1, P1, T2, P2, T3, P3, T4, P4, T5, P5),
not traitMax6(F, T1, P1, T2, P2, T3, P3, T4, P4, T5, P5, X, Y),
aPourMethodeAgregation(F,Ag).


%%%% Règles pour la construction des faits
estLieAkFonctionsfonctionsRestantes1(F, T1, P1)
:- estLieAServiceAvecPonderation(T1, F, P1),
```

```
estLieAServiceAvecPonderation(T2, F, P2), pred:inf(T1,T2).
fonctionMax1(F, T1, P1)
:- estLieAServiceAvecPonderation(T1, F, P1),
not fonctionsRestantes1(F, T1, P1).

fonctionsRestantes2(F, T1, P1)
:- estLieAServiceAvecPonderation(T1,F,P1),
estLieAServiceAvecPonderation(T2,F,P2),
pred:inf(T1,T2), fonctionMax1(F,T3,P3),
not pred:isEqualToString(T3,T2).
fonctionMax2(F, T1, P1, T2, P2)
:- fonctionMax1(F,T1,P1),
estLieAServiceAvecPonderation(T2,F,P2),
pred:inf(T2,T1),
not fonctionsRestantes2(F,T2,P2).

fonctionsRestantes3(F, T1, P1)
:- estLieAServiceAvecPonderation(T1,F,P1),
estLieAServiceAvecPonderation(T2,F,P2),
pred:inf(T1,T2), fonctionMax2(F,T3,P3,T4,P4),
not pred:isEqualToString(T3,T2),
not pred:isEqualToString(T4,T2).
fonctionMax3(F, T1, P1, T2, P2, T3, P3)
:- fonctionMax2(F,T1,P1,T2,P2),
estLieAServiceAvecPonderation(T3,F,P3),
not pred:isEqualToString(T1,T3),
not pred:isEqualToString(T2,T3),
not fonctionsRestantes3(F,T3,P3).

fonctionsRestantes4(F, T1, P1)
:- estLieAServiceAvecPonderation(T1,F,P1),
estLieAServiceAvecPonderation(T2,F,P2),
pred:inf(T1,T2), fonctionMax3(F,T3,P3,T4,P4,T5,P5),
not pred:isEqualToString(T3,T2),
not pred:isEqualToString(T4,T2),
not pred:isEqualToString(T5,T2).
fonctionMax4(F, T1, P1, T2, P2, T3, P3, T4, P4)
```

```
:- fonctionMax3(F,T1,P1,T2,P2,T3,P3),
estLieAServiceAvecPonderation(T4,F,P4),
not pred:isEqualToString(T1,T4),
not pred:isEqualToString(T2,T4),
not pred:isEqualToString(T3,T4),
not fonctionsRestantes4(F,T4,P4).

fonctionsRestantes5(F, T1, P1)
:- estLieAServiceAvecPonderation(T1,F,P1),
estLieAServiceAvecPonderation(T2,F,P2),
pred:inf(T1,T2),
fonctionMax4(F,T3,P3,T4,P4,T5,P5,T6,P6),
not pred:isEqualToString(T3,T2),
not pred:isEqualToString(T4,T2),
not pred:isEqualToString(T5,T2),
not pred:isEqualToString(T6,T2).
fonctionMax5(F, T1, P1, T2, P2, T3, P3, T4, P4, T5, P5)
:- fonctionMax4(F,T1,P1,T2,P2,T3,P3,T4,P4),
estLieAServiceAvecPonderation(T5,F,P5),
not pred:isEqualToString(T1,T5),
not pred:isEqualToString(T2,T5),
not pred:isEqualToString(T3,T5),
not pred:isEqualToString(T4,T5),
not fonctionsRestantes5(F,T5,P5).

estLieA1Fonction(F, T1, P1) :- fonctionMax1(F, T1, P1),
not fonctionMax2(F, T1, P1, X, Y).

estLieA2Fonctions(F, T1, P1, T2, P2, Ag)
:- fonctionMax2(F, T1, P1, T2, P2),
not fonctionMax3(F, T1, P1, T2, P2, X, Y),
aPourMethodeAgregation(F,Ag).

estLieA3Fonctions(F, T1, P1, T2, P2, T3, P3, Ag)
:- fonctionMax3(F, T1, P1, T2, P2, T3, P3),
not fonctionMax4(F, T1, P1, T2, P2, T3, P3, X, Y),
aPourMethodeAgregation(F,Ag).
```

```
estLieA4Fonctions(F, T1, P1, T2, P2, T3, P3, T4, P4, Ag)
:- fonctionMax4(F, T1, P1, T2, P2, T3, P3, T4, P4),
not fonctionMax5(F, T1, P1, T2, P2, T3, P3, T4, P4, X, Y),
aPourMethodeAgregation(F,Ag).

estLieA5Fonctions(F, T1, P1, T2, P2, T3, P3, T4, P4, T5, P5, Ag)
:- fonctionMax5(F, T1, P1, T2, P2, T3, P3, T4, P4, T5, P5),
not fonctionMax6(F, T1, P1, T2, P2, T3, P3, T4, P4, T5, P5, X, Y),
aPourMethodeAgregation(F,Ag).
```

Règles pour le passage des traits initiaux aux traits finaux

```
@computed fct:<./src/main/resources/functions.json>
@computed pred:<./src/main/resources/predicates.json>

@rules
nbrTrait(F,1) :- estLieA1Trait(F,T,S).
nbrTrait(F,2) :- estLieA2Traits(F,T1,S1,T2,S2,Ag).
nbrTrait(F,3) :- estLieA3Traits(F,T1,S1,T2,S2,T3,S3,Ag).
nbrTrait(F,4) :- estLieA4Traits(F,T1,S1,T2,S2,T3,S3,T4,S4,Ag).
nbrTrait(F,5) :- estLieA5Traits(F,T1,S1,T2,S2,T3,S3,T4,S4,T5,S5,Ag).

% Traits initiaux aux traits interchangeables
%% 1 trait correspondant dans une bd :
% CC = condition de croissance ; V = valeur ; BD = base de donnée
aPourValeurTraitBase(Trait,Esp,CC,V,BD):- apourespeceid(Esp,EspID,BD),
estLieA1TraitID(Trait,TraitID,BD),
apourvaleurtraitinitiale(TraitID,EspID,CC,V,BD).

%% 2 traits correspondants dans une même bd :
aPourValeurTraitBase(Trait,Esp,CC,fct:agreg(Ag,V1,1,V2,1),BD):-
estLieA2TraitIDs(Trait, TraitID1, TraitID2, BD, Ag),
apourvaleurtraitinitiale(TraitID1,EspID,CC,V1,BD),
apourvaleurtraitinitiale(TraitID2,EspID,CC,V2,BD),
apourespeceid(Esp,EspID,BD), not pred:isEqualToString(TraitID1,TraitID2).

aPourValeurTraitBase(Trait,Esp,CC,V1,BD):- apourespeceid(Esp,EspID,BD),
```

```
estLieA2TraitIDs(Trait, TraitID1, TraitID2, BD, Ag),
apourvaleurtraitinitiale(TraitID1,EspID,CC,V1,BD),
not apourvaleurtraitinitiale(TraitID2,EspID,CC,X,BD),
not pred:isEqualToString(TraitID1,TraitID2).

aPourValeurTraitBase(Trait,Esp,CC,V2,BD):- apourespeceid(Esp,EspID,BD),
estLieA2TraitIDs(Trait, TraitID1, TraitID2, BD, Ag),
not apourvaleurtraitinitiale(TraitID1,EspID,CC,V1,BD),
apourvaleurtraitinitiale(TraitID2,EspID,CC,V2,BD),
not pred:isEqualToString(TraitID1,TraitID2).

%% 3 traits correspondants dans une même bd :
aPourValeurTraitBase(Trait,Esp,CC,fct:agreg(Ag,V1,1,V2,1,V3,1),BD):-
apourespeceid(Esp,EspID,BD),
estLieA3TraitIDs(Trait, TraitID1, TraitID2, TraitID3, BD, Ag),
apourvaleurtraitinitiale(TraitID1,EspID,CC,V1,BD),
apourvaleurtraitinitiale(TraitID2,EspID,CC,V2,BD),
apourvaleurtraitinitiale(TraitID3,EspID,CC,V3,BD),
not pred:isEqualToString(TraitID1,TraitID2),
not pred:isEqualToString(TraitID2,TraitID3),
not pred:isEqualToString(TraitID1,TraitID3).

aPourValeurTraitBase(Trait,Esp,CC,fct:agreg(Ag,V1,1,V2,1),BD):-
apourespeceid(Esp,EspID,BD),
estLieA3TraitIDs(Trait, TraitID1, TraitID2, TraitID3, BD, Ag),
apourvaleurtraitinitiale(TraitID1,EspID,CC,V1,BD),
apourvaleurtraitinitiale(TraitID2,EspID,CC,V2,BD),
not apourvaleurtraitinitiale(TraitID3,EspID,CC,X,BD),
not pred:isEqualToString(TraitID1,TraitID2),
not pred:isEqualToString(TraitID2,TraitID3),
not pred:isEqualToString(TraitID1,TraitID3).

aPourValeurTraitBase(Trait,Esp,CC,fct:agreg(Ag,V1,1,V3,1),BD):-
apourespeceid(Esp,EspID,BD),
estLieA3TraitIDs(Trait, TraitID1, TraitID2, TraitID3, BD, Ag),
apourvaleurtraitinitiale(TraitID1,EspID,CC,V1,BD),
not apourvaleurtraitinitiale(TraitID2,EspID,CC,V2,BD),
```

```
apourvaleurtraitinitiale(TraitID3,EspID,CC,V3,BD),
not pred:isEqualToString(TraitID1,TraitID2),
not pred:isEqualToString(TraitID2,TraitID3),
not pred:isEqualToString(TraitID1,TraitID3).

aPourValeurTraitBase(Trait,Esp,CC,fct:agreg(Ag,V2,1,V3,1),BD):-
apourespeceid(Esp,EspID,BD),
estLieA3TraitIDs(Trait, TraitID1, TraitID2, TraitID3, BD, Ag),
not apourvaleurtraitinitiale(TraitID1,EspID,CC,V1,BD),
apourvaleurtraitinitiale(TraitID2,EspID,CC,V2,BD),
apourvaleurtraitinitiale(TraitID3,EspID,CC,V3,BD),
not pred:isEqualToString(TraitID1,TraitID2),
not pred:isEqualToString(TraitID2,TraitID3),
not pred:isEqualToString(TraitID1,TraitID3).

aPourValeurTraitBase(Trait,Esp,CC,V3,BD):- apourespeceid(Esp,EspID,BD),
estLieA3TraitIDs(Trait, TraitID1, TraitID2, TraitID3, BD, Ag),
not apourvaleurtraitinitiale(TraitID1,EspID,CC,V1,BD),
not apourvaleurtraitinitiale(TraitID2,EspID,CC,V2,BD),
apourvaleurtraitinitiale(TraitID3,EspID,CC,V3,BD),
not pred:isEqualToString(TraitID1,TraitID2),
not pred:isEqualToString(TraitID2,TraitID3),
not pred:isEqualToString(TraitID1,TraitID3).

aPourValeurTraitBase(Trait,Esp,CC,V2,BD):- apourespeceid(Esp,EspID,BD),
estLieA3TraitIDs(Trait, TraitID1, TraitID2, TraitID3, BD, Ag),
not apourvaleurtraitinitiale(TraitID1,EspID,CC,V1,BD),
apourvaleurtraitinitiale(TraitID2,EspID,CC,V2,BD),
not apourvaleurtraitinitiale(TraitID3,EspID,CC,V3,BD),
not pred:isEqualToString(TraitID1,TraitID2),
not pred:isEqualToString(TraitID2,TraitID3),
not pred:isEqualToString(TraitID1,TraitID3).

aPourValeurTraitBase(Trait,Esp,CC,V1,BD):- apourespeceid(Esp,EspID,BD),
estLieA3TraitIDs(Trait, TraitID1, TraitID2, TraitID3, BD, Ag),
apourvaleurtraitinitiale(TraitID1,EspID,CC,V1,BD),
not apourvaleurtraitinitiale(TraitID2,EspID,CC,V2,BD),
```

```
not apourvaleurtraitinitiale(TraitID3,EspID,CC,V3,BD),
not pred:isEqualToString(TraitID1,TraitID2),
not pred:isEqualToString(TraitID2,TraitID3),
not pred:isEqualToString(TraitID1,TraitID3).

%% 4 traits correspondants dans une même bd :
aPourValeurTraitBase(Trait,Esp,CC,fct:agreg(Ag,V1,1,V2,1,V3,1,V4,1),BD):-
apourespeceid(Esp,EspID,BD),
estLieA3TraitIDs(Trait, TraitID1, TraitID2, TraitID3, TraitID4, BD, Ag),
apourvaleurtraitinitiale(TraitID1,EspID,CC,V1,BD),
apourvaleurtraitinitiale(TraitID2,EspID,CC,V2,BD),
apourvaleurtraitinitiale(TraitID3,EspID,CC,V3,BD),
apourvaleurtraitinitiale(TraitID4,EspID,CC,V4,BD),
not pred:isEqualToString(TraitID1,TraitID2),
not pred:isEqualToString(TraitID2,TraitID3),
not pred:isEqualToString(TraitID3,TraitID4),
not pred:isEqualToString(TraitID1,TraitID3),
not pred:isEqualToString(TraitID1,TraitID4),
not pred:isEqualToString(TraitID2,TraitID4).

aPourValeurTraitBase(Trait,Esp,CC,fct:agreg(Ag,V2,1,V3,1,V4,1),BD):-
apourespeceid(Esp,EspID,BD),
estLieA3TraitIDs(Trait, TraitID1, TraitID2, TraitID3, TraitID4, BD, Ag),
not apourvaleurtraitinitiale(TraitID1,EspID,CC,V1,BD),
apourvaleurtraitinitiale(TraitID2,EspID,CC,V2,BD),
apourvaleurtraitinitiale(TraitID3,EspID,CC,V3,BD),
apourvaleurtraitinitiale(TraitID4,EspID,CC,V4,BD),
not pred:isEqualToString(TraitID1,TraitID2),
not pred:isEqualToString(TraitID2,TraitID3),
not pred:isEqualToString(TraitID3,TraitID4),
not pred:isEqualToString(TraitID1,TraitID3),
not pred:isEqualToString(TraitID1,TraitID4),
not pred:isEqualToString(TraitID2,TraitID4).

aPourValeurTraitBase(Trait,Esp,CC,fct:agreg(Ag,V1,1,V3,1,V4,1),BD):-
apourespeceid(Esp,EspID,BD),
estLieA3TraitIDs(Trait, TraitID1, TraitID2, TraitID3, TraitID4, BD, Ag),
```

```
apourvaleurtraitinitiale(TraitID1,EspID,CC,V1,BD),
not apourvaleurtraitinitiale(TraitID2,EspID,CC,V2,BD),
apourvaleurtraitinitiale(TraitID3,EspID,CC,V3,BD),
apourvaleurtraitinitiale(TraitID4,EspID,CC,V4,BD),
not pred:isEqualToString(TraitID1,TraitID2),
not pred:isEqualToString(TraitID2,TraitID3),
not pred:isEqualToString(TraitID3,TraitID4),
not pred:isEqualToString(TraitID1,TraitID3),
not pred:isEqualToString(TraitID1,TraitID4),
not pred:isEqualToString(TraitID2,TraitID4).

aPourValeurTraitBase(Trait,Esp,CC,fct:agreg(Ag,V1,1,V2,1,V4,1),BD):-
apourespeceid(Esp,EspID,BD),
estLieA3TraitIDs(Trait, TraitID1, TraitID2, TraitID3, TraitID4, BD, Ag),
apourvaleurtraitinitiale(TraitID1,EspID,CC,V1,BD),
apourvaleurtraitinitiale(TraitID2,EspID,CC,V2,BD),
not apourvaleurtraitinitiale(TraitID3,EspID,CC,V3,BD),
apourvaleurtraitinitiale(TraitID4,EspID,CC,V4,BD),
not pred:isEqualToString(TraitID1,TraitID2),
not pred:isEqualToString(TraitID2,TraitID3),
not pred:isEqualToString(TraitID3,TraitID4),
not pred:isEqualToString(TraitID1,TraitID3),
not pred:isEqualToString(TraitID1,TraitID4),
not pred:isEqualToString(TraitID2,TraitID4).

aPourValeurTraitBase(Trait,Esp,CC,fct:agreg(Ag,V1,1,V2,1,V3,1),BD):-
apourespeceid(Esp,EspID,BD),
estLieA3TraitIDs(Trait, TraitID1, TraitID2, TraitID3, TraitID4, BD, Ag),
apourvaleurtraitinitiale(TraitID1,EspID,CC,V1,BD),
apourvaleurtraitinitiale(TraitID2,EspID,CC,V2,BD),
apourvaleurtraitinitiale(TraitID3,EspID,CC,V3,BD),
not apourvaleurtraitinitiale(TraitID4,EspID,CC,V4,BD),
not pred:isEqualToString(TraitID1,TraitID2),
not pred:isEqualToString(TraitID2,TraitID3),
not pred:isEqualToString(TraitID3,TraitID4),
not pred:isEqualToString(TraitID1,TraitID3),
not pred:isEqualToString(TraitID1,TraitID4),
```

```
not pred:isEqualToString(TraitID2,TraitID4).

aPourValeurTraitBase(Trait,Esp,CC,fct:agreg(Ag,V3,1,V4,1),BD):-
apourespeceid(Esp,EspID,BD),
estLieA3TraitIDs(Trait, TraitID1, TraitID2, TraitID3, TraitID4, BD, Ag),
not apourvaleurtraitinitiale(TraitID1,EspID,CC,V1,BD),
not apourvaleurtraitinitiale(TraitID2,EspID,CC,V2,BD),
apourvaleurtraitinitiale(TraitID3,EspID,CC,V3,BD),
apourvaleurtraitinitiale(TraitID4,EspID,CC,V4,BD),
not pred:isEqualToString(TraitID1,TraitID2),
not pred:isEqualToString(TraitID2,TraitID3),
not pred:isEqualToString(TraitID3,TraitID4),
not pred:isEqualToString(TraitID1,TraitID3),
not pred:isEqualToString(TraitID1,TraitID4),
not pred:isEqualToString(TraitID2,TraitID4).

aPourValeurTraitBase(Trait,Esp,CC,fct:agreg(Ag,V2,1,V4,1),BD):-
apourespeceid(Esp,EspID,BD),
estLieA3TraitIDs(Trait, TraitID1, TraitID2, TraitID3, TraitID4, BD, Ag),
not apourvaleurtraitinitiale(TraitID1,EspID,CC,V1,BD),
apourvaleurtraitinitiale(TraitID2,EspID,CC,V2,BD),
not apourvaleurtraitinitiale(TraitID3,EspID,CC,V3,BD),
apourvaleurtraitinitiale(TraitID4,EspID,CC,V4,BD),
not pred:isEqualToString(TraitID1,TraitID2),
not pred:isEqualToString(TraitID2,TraitID3),
not pred:isEqualToString(TraitID3,TraitID4),
not pred:isEqualToString(TraitID1,TraitID3),
not pred:isEqualToString(TraitID1,TraitID4),
not pred:isEqualToString(TraitID2,TraitID4).

aPourValeurTraitBase(Trait,Esp,CC,fct:agreg(Ag,V2,1,V3,1),BD):-
apourespeceid(Esp,EspID,BD),
estLieA3TraitIDs(Trait, TraitID1, TraitID2, TraitID3, TraitID4, BD, Ag),
not apourvaleurtraitinitiale(TraitID1,EspID,CC,V1,BD),
apourvaleurtraitinitiale(TraitID2,EspID,CC,V2,BD),
apourvaleurtraitinitiale(TraitID3,EspID,CC,V3,BD),
not apourvaleurtraitinitiale(TraitID4,EspID,CC,V4,BD),
```

```
not pred:isEqualToString(TraitID1,TraitID2),
not pred:isEqualToString(TraitID2,TraitID3),
not pred:isEqualToString(TraitID3,TraitID4),
not pred:isEqualToString(TraitID1,TraitID3),
not pred:isEqualToString(TraitID1,TraitID4),
not pred:isEqualToString(TraitID2,TraitID4).

aPourValeurTraitBase(Trait,Esp,CC,fct:agreg(Ag,V1,1,V4,1),BD):-
apourespeceid(Esp,EspID,BD),
estLieA3TraitIDs(Trait, TraitID1, TraitID2, TraitID3, TraitID4, BD, Ag),
apourvaleurtraitinitiale(TraitID1,EspID,CC,V1,BD),
not apourvaleurtraitinitiale(TraitID2,EspID,CC,V2,BD),
not apourvaleurtraitinitiale(TraitID3,EspID,CC,V3,BD),
apourvaleurtraitinitiale(TraitID4,EspID,CC,V4,BD),
not pred:isEqualToString(TraitID1,TraitID2),
not pred:isEqualToString(TraitID2,TraitID3),
not pred:isEqualToString(TraitID3,TraitID4),
not pred:isEqualToString(TraitID1,TraitID3),
not pred:isEqualToString(TraitID1,TraitID4),
not pred:isEqualToString(TraitID2,TraitID4).

aPourValeurTraitBase(Trait,Esp,CC,fct:agreg(Ag,V1,1,V3,1),BD):-
apourespeceid(Esp,EspID,BD),
estLieA3TraitIDs(Trait, TraitID1, TraitID2, TraitID3, TraitID4, BD, Ag),
apourvaleurtraitinitiale(TraitID1,EspID,CC,V1,BD),
not apourvaleurtraitinitiale(TraitID2,EspID,CC,V2,BD),
apourvaleurtraitinitiale(TraitID3,EspID,CC,V3,BD),
not apourvaleurtraitinitiale(TraitID4,EspID,CC,V4,BD),
not pred:isEqualToString(TraitID1,TraitID2),
not pred:isEqualToString(TraitID2,TraitID3),
not pred:isEqualToString(TraitID3,TraitID4),
not pred:isEqualToString(TraitID1,TraitID3),
not pred:isEqualToString(TraitID1,TraitID4),
not pred:isEqualToString(TraitID2,TraitID4).

aPourValeurTraitBase(Trait,Esp,CC,fct:agreg(Ag,V1,1,V2,1),BD):-
apourespeceid(Esp,EspID,BD),
```

```
estLieA3TraitIDs(Trait, TraitID1, TraitID2, TraitID3, TraitID4, BD, Ag),
apourvaleurtraitinitiale(TraitID1,EspID,CC,V1,BD),
apourvaleurtraitinitiale(TraitID2,EspID,CC,V2,BD),
not apourvaleurtraitinitiale(TraitID3,EspID,CC,V3,BD),
not apourvaleurtraitinitiale(TraitID4,EspID,CC,V4,BD),
not pred:isEqualToString(TraitID1,TraitID2),
not pred:isEqualToString(TraitID2,TraitID3),
not pred:isEqualToString(TraitID3,TraitID4),
not pred:isEqualToString(TraitID1,TraitID3),
not pred:isEqualToString(TraitID1,TraitID4),
not pred:isEqualToString(TraitID2,TraitID4).


aPourValeurTraitBase(Trait,Esp,CC,V1,BD):-
apourespeceid(Esp,EspID,BD),
estLieA3TraitIDs(Trait, TraitID1, TraitID2, TraitID3, TraitID4, BD, Ag),
apourvaleurtraitinitiale(TraitID1,EspID,CC,V1,BD),
not apourvaleurtraitinitiale(TraitID2,EspID,CC,V2,BD),
not apourvaleurtraitinitiale(TraitID3,EspID,CC,V3,BD),
not apourvaleurtraitinitiale(TraitID4,EspID,CC,V4,BD),
not pred:isEqualToString(TraitID1,TraitID2),
not pred:isEqualToString(TraitID2,TraitID3),
not pred:isEqualToString(TraitID3,TraitID4),
not pred:isEqualToString(TraitID1,TraitID3),
not pred:isEqualToString(TraitID1,TraitID4),
not pred:isEqualToString(TraitID2,TraitID4).


aPourValeurTraitBase(Trait,Esp,CC,V2,BD):-
apourespeceid(Esp,EspID,BD),
estLieA3TraitIDs(Trait, TraitID1, TraitID2, TraitID3, TraitID4, BD, Ag),
not apourvaleurtraitinitiale(TraitID1,EspID,CC,V1,BD),
apourvaleurtraitinitiale(TraitID2,EspID,CC,V2,BD),
not apourvaleurtraitinitiale(TraitID3,EspID,CC,V3,BD),
not apourvaleurtraitinitiale(TraitID4,EspID,CC,V4,BD),
not pred:isEqualToString(TraitID1,TraitID2),
not pred:isEqualToString(TraitID2,TraitID3),
```

```
not pred:isEqualToString(TraitID3,TraitID4),
not pred:isEqualToString(TraitID1,TraitID3),
not pred:isEqualToString(TraitID1,TraitID4),
not pred:isEqualToString(TraitID2,TraitID4).

aPourValeurTraitBase(Trait,Esp,CC,V3,BD):-
apourespeceid(Esp,EspID,BD),
estLieA3TraitIDs(Trait, TraitID1, TraitID2, TraitID3, TraitID4, BD, Ag),
not apourvaleurtraitinitiale(TraitID1,EspID,CC,V1,BD),
not apourvaleurtraitinitiale(TraitID2,EspID,CC,V2,BD),
apourvaleurtraitinitiale(TraitID3,EspID,CC,V3,BD),
not apourvaleurtraitinitiale(TraitID4,EspID,CC,V4,BD),
not pred:isEqualToString(TraitID1,TraitID2),
not pred:isEqualToString(TraitID2,TraitID3),
not pred:isEqualToString(TraitID3,TraitID4),
not pred:isEqualToString(TraitID1,TraitID3),
not pred:isEqualToString(TraitID1,TraitID4),
not pred:isEqualToString(TraitID2,TraitID4).

aPourValeurTraitBase(Trait,Esp,CC,V4,BD):-
apourespeceid(Esp,EspID,BD),
estLieA3TraitIDs(Trait, TraitID1, TraitID2, TraitID3, TraitID4, BD, Ag),
not apourvaleurtraitinitiale(TraitID1,EspID,CC,V1,BD),
not apourvaleurtraitinitiale(TraitID2,EspID,CC,V2,BD),
not apourvaleurtraitinitiale(TraitID3,EspID,CC,V3,BD),
apourvaleurtraitinitiale(TraitID4,EspID,CC,V4,BD),
not pred:isEqualToString(TraitID1,TraitID2),
not pred:isEqualToString(TraitID2,TraitID3),
not pred:isEqualToString(TraitID3,TraitID4),
not pred:isEqualToString(TraitID1,TraitID3),
not pred:isEqualToString(TraitID1,TraitID4),
not pred:isEqualToString(TraitID2,TraitID4).


%%%% Gestion des priorités entre les bases; définition d'Atomes "prioritaires"
valeurTraitPrioritaireSur(Trait,Esp,V1,B1,V2,B2,CC) :-
aPourValeurTraitBase(Trait,Esp,CC,V1,B1),
```

```
aPourValeurTraitBase(Trait,Esp,CC,V2,B2), traitPrioritaireBase(Trait,B1,B2).

valeurTraitPrioritaireSur(Trait,Esp,V1,B1,V2,B2,CC) :-
aPourValeurTraitBase(Trait,Esp,CC,V1,B1),
aPourValeurTraitBase(Trait,Esp,CC,V2,B2),
basePrioritaire(B1,B2), not traitPrioritaireBase(Trait,B2,B1).

% Traits interchangeables aux traits finaux
aPourValeurTraitPrioritaire(Trait,Esp,V2,B2,CC) :-
valeurTraitPrioritaireSur(Trait,Esp,V1,B1,V2,B2,CC).

aPourValeurTrait(Trait,Esp,V,CC) :- aPourValeurTraitBase(Trait,Esp,CC,V,BD),
not aPourValeurTraitPrioritaire(Trait,Esp,V,BD,CC).
```

Règles pour le calcul de valeurs de fonctions, puis de services

```
@computed fct:<./src/main/resources/functions.json>
@computed pred:<./src/main/resources/predicates.json>

@rules
% Évaluation d'une fonction par rapport aux valeurs des traits liés
% pour une espèce donnée
%%%% Fonction liée à un trait
aPourValeurFonctionCalculee(Fct,Esp,fct:une_valeur(V,S),100,CC) :-
estLieA1Trait(Fct, Trait, S),  aPourValeurTrait(Trait,Esp,V,CC).

%%%% Fonction liée à 2 traits
aPourValeurFonctionCalculee(Fct,Esp,fct:agreg(Ag,V1,S1,V2,S2),100,CC) :-
estLieA2Traits(Fct,T1,S1,T2,S2,Ag),
aPourValeurTrait(T1,Esp,V1,CC), aPourValeurTrait(T2,Esp,V2,CC).

aPourValeurFonctionCalculee(Fct,Esp,fct:une_valeur(V2,S2),50,CC) :-
estLieA2Traits(Fct,T1,S1,T2,S2,Ag),
not aPourValeurTrait(T1,Esp,V1,CC), aPourValeurTrait(T2,Esp,V2,CC).

aPourValeurFonctionCalculee(Fct,Esp,fct:une_valeur(V1,S1),50,CC) :-
estLieA2Traits(Fct,T1,S1,T2,S2,Ag),
```

```
aPourValeurTrait(T1,Esp,V1,CC), not aPourValeurTrait(T2,Esp,X,CC).


%%%% Fonction liée à 3 traits
aPourValeurFonctionCalculee(Fct,Esp,fct:agreg(Ag,V1,S1,V2,S2,V3,S3),100,CC) :-
estLieA3Traits(Fct,T1,S1,T2,S2,T3,S3,Ag), aPourValeurTrait(T1,Esp,V1,CC),
aPourValeurTrait(T2,Esp,V2,CC), aPourValeurTrait(T3,Esp,V3,CC).

aPourValeurFonctionCalculee(Fct,Esp,fct:agreg(Ag,V1,S1,V2,S2),66,
CC) :- estLieA3Traits(Fct,T1,S1,T2,S2,T3,S3,Ag),
aPourValeurTrait(T1,Esp,V1,CC),
aPourValeurTrait(T2,Esp,V2,CC), not aPourValeurTrait(T3,Esp,X,CC).

aPourValeurFonctionCalculee(Fct,Esp,fct:agreg(Ag,V1,S1,V3,S3),66,
CC) :- estLieA3Traits(Fct,T1,S1,T2,S2,T3,S3,Ag),
aPourValeurTrait(T1,Esp,V1,CC),
not aPourValeurTrait(T2,Esp,V2,CC), aPourValeurTrait(T3,Esp,V3,CC).

aPourValeurFonctionCalculee(Fct,Esp,fct:agreg(Ag,V2,S2,V3,S3),66,
CC) :- estLieA3Traits(Fct,T1,S1,T2,S2,T3,S3,Ag),
not aPourValeurTrait(T1,Esp,V1,CC),
aPourValeurTrait(T2,Esp,V2,CC), aPourValeurTrait(T3,Esp,V3,CC).

aPourValeurFonctionCalculee(Fct,Esp,fct:une_valeur(V1,S1),33,CC) :-
estLieA3Traits(Fct,T1,S1,T2,S2,T3,S3,Ag),
aPourValeurTrait(T1,Esp,V1,CC),
not aPourValeurTrait(T2,Esp,X,CC), not aPourValeurTrait(T3,Esp,X,CC).

aPourValeurFonctionCalculee(Fct,Esp,fct:une_valeur(V2,S2),33,CC) :-
estLieA3Traits(Fct,T1,S1,T2,S2,T3,S3,Ag),
not aPourValeurTrait(T1,Esp,V1,CC),
aPourValeurTrait(T2,Esp,V2,CC), not aPourValeurTrait(T3,Esp,V3,CC).

aPourValeurFonctionCalculee(Fct,Esp,fct:une_valeur(V3,S3),33,CC) :-
estLieA3Traits(Fct,T1,S1,T2,S2,T3,S3,Ag),
not aPourValeurTrait(T1,Esp,V1,CC),
not aPourValeurTrait(T2,Esp,V2,CC), aPourValeurTrait(T3,Esp,V3,CC).
```

```
%%% Fonction liée à 4 traits
aPourValeurFonctionCalculee(Fct,Esp,fct:agreg(Ag,V1,S1,V2,S2,
V3,S3,V4,S4),100,CC) :-
estLieA4Traits(Fct,T1,S1,T2,S2,T3,S3,T4,S4,Ag),
aPourValeurTrait(T1,Esp,V1,CC), aPourValeurTrait(T2,Esp,V2,CC),
aPourValeurTrait(T3,Esp,V3,CC), aPourValeurTrait(T4,Esp,V4,CC).

aPourValeurFonctionCalculee(Fct,Esp,fct:agreg(Ag,V1,S1,V2,S2,V3,S3),
75,CC) :- estLieA4Traits(Fct,T1,S1,T2,S2,T3,S3,T4,S4,Ag),
aPourValeurTrait(T1,Esp,V1,CC), aPourValeurTrait(T2,Esp,V2,CC),
aPourValeurTrait(T3,Esp,V3,CC), not aPourValeurTrait(T4,Esp,X,CC).

aPourValeurFonctionCalculee(Fct,Esp,fct:agreg(Ag,V1,S1,V2,S2,V4,S4),
75,CC) :- estLieA4Traits(Fct,T1,S1,T2,S2,T3,S3,T4,S4,Ag),
aPourValeurTrait(T1,Esp,V1,CC), aPourValeurTrait(T2,Esp,V2,CC),
not aPourValeurTrait(T3,Esp,V3,CC), aPourValeurTrait(T4,Esp,V4,CC).

aPourValeurFonctionCalculee(Fct,Esp,fct:agreg(Ag,V1,S1,V3,S3,V4,S4),
75,CC) :- estLieA4Traits(Fct,T1,S1,T2,S2,T3,S3,T4,S4,Ag),
aPourValeurTrait(T1,Esp,V1,CC),not aPourValeurTrait(T2,Esp,V2,CC),
aPourValeurTrait(T3,Esp,V3,CC), aPourValeurTrait(T4,Esp,V4,CC).

aPourValeurFonctionCalculee(Fct,Esp,fct:agreg(Ag,V2,S2,V3,S3,V4,S4),
75,CC) :- estLieA4Traits(Fct,T1,S1,T2,S2,T3,S3,T4,S4,Ag),
not aPourValeurTrait(T1,Esp,V1,CC), aPourValeurTrait(T2,Esp,V2,CC),
aPourValeurTrait(T3,Esp,V3,CC), aPourValeurTrait(T4,Esp,V4,CC).

aPourValeurFonctionCalculee(Fct,Esp,fct:agreg(Ag,V3,S3,V4,S4),50,CC)
:- estLieA4Traits(Fct,T1,S1,T2,S2,T3,S3,T4,S4,Ag),
not aPourValeurTrait(T1,Esp,V1,CC), not aPourValeurTrait(T2,Esp,V2,CC),
aPourValeurTrait(T3,Esp,V3,CC), aPourValeurTrait(T4,Esp,V4,CC).

aPourValeurFonctionCalculee(Fct,Esp,fct:agreg(Ag,V2,S2,V4,S4),50,CC)
:- estLieA4Traits(Fct,T1,S1,T2,S2,T3,S3,T4,S4,Ag),
not aPourValeurTrait(T1,Esp,V1,CC),  aPourValeurTrait(T2,Esp,V2,CC),
not aPourValeurTrait(T3,Esp,V3,CC), aPourValeurTrait(T4,Esp,V4,CC).
```

```
aPourValeurFonctionCalculee(Fct,Esp,fct:agreg(Ag,V2,S2,V3,S3),50,CC)
:- estLieA4Traits(Fct,T1,S1,T2,S2,T3,S3,T4,S4,Ag),
not aPourValeurTrait(T1,Esp,V1,CC),  aPourValeurTrait(T2,Esp,V2,CC),
aPourValeurTrait(T3,Esp,V3,CC), not aPourValeurTrait(T4,Esp,V4,CC).

aPourValeurFonctionCalculee(Fct,Esp,fct:agreg(Ag,V1,S1,V4,S4),50,CC)
:- estLieA4Traits(Fct,T1,S1,T2,S2,T3,S3,T4,S4,Ag),
 aPourValeurTrait(T1,Esp,V1,CC), not aPourValeurTrait(T2,Esp,V2,CC),
 not aPourValeurTrait(T3,Esp,V3,CC), aPourValeurTrait(T4,Esp,V4,CC).

aPourValeurFonctionCalculee(Fct,Esp,fct:agreg(Ag,V1,S1,V3,S3),50,CC)
:- estLieA4Traits(Fct,T1,S1,T2,S2,T3,S3,T4,S4,Ag),
 aPourValeurTrait(T1,Esp,V1,CC), not aPourValeurTrait(T2,Esp,V2,CC),
 aPourValeurTrait(T3,Esp,V3,CC), not aPourValeurTrait(T4,Esp,V4,CC).

aPourValeurFonctionCalculee(Fct,Esp,fct:agreg(Ag,V1,S1,V2,S2),50,CC)
:- estLieA4Traits(Fct,T1,S1,T2,S2,T3,S3,T4,S4,Ag),
 aPourValeurTrait(T1,Esp,V1,CC),  aPourValeurTrait(T2,Esp,V2,CC),
 not aPourValeurTrait(T3,Esp,V3,CC), not aPourValeurTrait(T4,Esp,V4,CC).

aPourValeurFonctionCalculee(Fct,Esp,fct:une_valeur(V1,S1),25,CC)
:- estLieA4Traits(Fct,T1,S1,T2,S2,T3,S3,T4,S4,Ag),
 aPourValeurTrait(T1,Esp,V1,CC), not aPourValeurTrait(T2,Esp,V2,CC),
 not aPourValeurTrait(T3,Esp,V3,CC), not aPourValeurTrait(T4,Esp,V4,CC).

aPourValeurFonctionCalculee(Fct,Esp,fct:une_valeur(V2,S2),25,CC)
:- estLieA4Traits(Fct,T1,S1,T2,S2,T3,S3,T4,S4,Ag),
not aPourValeurTrait(T1,Esp,V1,CC),  aPourValeurTrait(T2,Esp,V2,CC),
not aPourValeurTrait(T3,Esp,V3,CC), not aPourValeurTrait(T4,Esp,V4,CC).

aPourValeurFonctionCalculee(Fct,Esp,fct:une_valeur(V3,S3),25,CC)
:- estLieA4Traits(Fct,T1,S1,T2,S2,T3,S3,T4,S4,Ag),
not aPourValeurTrait(T1,Esp,V1,CC), not aPourValeurTrait(T2,Esp,V2,CC),
aPourValeurTrait(T3,Esp,V3,CC), not aPourValeurTrait(T4,Esp,V4,CC).

aPourValeurFonctionCalculee(Fct,Esp,fct:une_valeur(V4,S4),25,CC)
```

```
:- estLieA4Traits(Fct,T1,S1,T2,S2,T3,S3,T4,S4,Ag),
not aPourValeurTrait(T1,Esp,V1,CC), not aPourValeurTrait(T2,Esp,V2,CC),
not aPourValeurTrait(T3,Esp,V3,CC), aPourValeurTrait(T4,Esp,V4,CC).


%%%% Fonction liée à 5 traits
aPourValeurFonctionCalculee(Fct,Esp,
fct:agreg(Ag,V1,S1,V2,S2,V3,S3,V4,S4,V5,S5),100,CC) :-
estLieA5Traits(Fct,T1,S1,T2,S2,T3,S3,T4,S4,T5,S5,Ag),
aPourValeurTrait(T1,Esp,V1,CC), aPourValeurTrait(T2,Esp,V2,CC),
aPourValeurTrait(T3,Esp,V3,CC), aPourValeurTrait(T4,Esp,V4,CC),
aPourValeurTrait(T5,Esp,V5,CC).

aPourValeurFonctionCalculee(Fct,Esp,
fct:agreg(Ag,V1,S1,V2,S2,V3,S3,V4,S4),80,CC) :-
estLieA5Traits(Fct,T1,S1,T2,S2,T3,S3,T4,S4,T5,S5,Ag),
aPourValeurTrait(T1,Esp,V1,CC), aPourValeurTrait(T2,Esp,V2,CC),
aPourValeurTrait(T3,Esp,V3,CC), aPourValeurTrait(T4,Esp,V4,CC),
not aPourValeurTrait(T5,Esp,V5,CC).

aPourValeurFonctionCalculee(Fct,Esp,
fct:agreg(Ag,V1,S1,V2,S2,V3,S3,V5,S5),80,CC) :-
estLieA5Traits(Fct,T1,S1,T2,S2,T3,S3,T4,S4,T5,S5,Ag),
aPourValeurTrait(T1,Esp,V1,CC), aPourValeurTrait(T2,Esp,V2,CC),
aPourValeurTrait(T3,Esp,V3,CC), not aPourValeurTrait(T4,Esp,V4,CC),
 aPourValeurTrait(T5,Esp,V5,CC).

aPourValeurFonctionCalculee(Fct,Esp,
fct:agreg(Ag,V1,S1,V2,S2,V4,S4,V5,S5),80,CC) :-
estLieA5Traits(Fct,T1,S1,T2,S2,T3,S3,T4,S4,T5,S5,Ag),
aPourValeurTrait(T1,Esp,V1,CC), aPourValeurTrait(T2,Esp,V2,CC),
not aPourValeurTrait(T3,Esp,V3,CC), aPourValeurTrait(T4,Esp,V4,CC),
 aPourValeurTrait(T5,Esp,V5,CC).

aPourValeurFonctionCalculee(Fct,Esp,
fct:agreg(Ag,V1,S1,V3,S3,V4,S4,V5,S5),80,CC) :-
 estLieA5Traits(Fct,T1,S1,T2,S2,T3,S3,T4,S4,T5,S5,Ag),
```

```
aPourValeurTrait(T1,Esp,V1,CC), not aPourValeurTrait(T2,Esp,V2,CC),
aPourValeurTrait(T3,Esp,V3,CC), aPourValeurTrait(T4,Esp,V4,CC),
 aPourValeurTrait(T5,Esp,V5,CC).

aPourValeurFonctionCalculee(Fct,Esp,
fct:agreg(Ag,V2,S2,V3,S3,V4,S4,V5,S5),80,CC) :-
estLieA5Traits(Fct,T1,S1,T2,S2,T3,S3,T4,S4,T5,S5,Ag),
not aPourValeurTrait(T1,Esp,V1,CC), aPourValeurTrait(T2,Esp,V2,CC),
aPourValeurTrait(T3,Esp,V3,CC), aPourValeurTrait(T4,Esp,V4,CC),
 aPourValeurTrait(T5,Esp,V5,CC).

aPourValeurFonctionCalculee(Fct,Esp,fct:agreg(Ag,V3,S3,V4,S4,V5,S5),60,CC)
:- estLieA5Traits(Fct,T1,S1,T2,S2,T3,S3,T4,S4,T5,S5,Ag),
not aPourValeurTrait(T1,Esp,V1,CC), not aPourValeurTrait(T2,Esp,V2,CC),
aPourValeurTrait(T3,Esp,V3,CC), aPourValeurTrait(T4,Esp,V4,CC),
aPourValeurTrait(T5,Esp,V5,CC).

aPourValeurFonctionCalculee(Fct,Esp,fct:agreg(Ag,V2,S2,V4,S4,V5,S5),60,CC)
:- estLieA5Traits(Fct,T1,S1,T2,S2,T3,S3,T4,S4,T5,S5,Ag),
not aPourValeurTrait(T1,Esp,V1,CC), aPourValeurTrait(T2,Esp,V2,CC),
not aPourValeurTrait(T3,Esp,V3,CC), aPourValeurTrait(T4,Esp,V4,CC),
aPourValeurTrait(T5,Esp,V5,CC).

aPourValeurFonctionCalculee(Fct,Esp,fct:agreg(Ag,V2,S2,V3,S3,V5,S5),60,CC)
:- estLieA5Traits(Fct,T1,S1,T2,S2,T3,S3,T4,S4,T5,S5,Ag),
not aPourValeurTrait(T1,Esp,V1,CC), aPourValeurTrait(T2,Esp,V2,CC),
aPourValeurTrait(T3,Esp,V3,CC),not aPourValeurTrait(T4,Esp,V4,CC),
aPourValeurTrait(T5,Esp,V5,CC).

aPourValeurFonctionCalculee(Fct,Esp,fct:agreg(Ag,V2,S2,V3,S3,V4,S4),60,CC)
:- estLieA5Traits(Fct,T1,S1,T2,S2,T3,S3,T4,S4,T5,S5,Ag),
not aPourValeurTrait(T1,Esp,V1,CC), aPourValeurTrait(T2,Esp,V2,CC),
aPourValeurTrait(T3,Esp,V3,CC), aPourValeurTrait(T4,Esp,V4,CC),
not aPourValeurTrait(T5,Esp,V5,CC).

aPourValeurFonctionCalculee(Fct,Esp,fct:agreg(Ag,V1,S1,V4,S4,V5,S5),60,CC)
:- estLieA5Traits(Fct,T1,S1,T2,S2,T3,S3,T4,S4,T5,S5,Ag),
```

```
aPourValeurTrait(T1,Esp,V1,CC), not aPourValeurTrait(T2,Esp,V2,CC),
not aPourValeurTrait(T3,Esp,V3,CC), aPourValeurTrait(T4,Esp,V4,CC),
aPourValeurTrait(T5,Esp,V5,CC).

aPourValeurFonctionCalculee(Fct,Esp,fct:agreg(Ag,V1,S1,V3,S3,V5,S5),60,CC)
:- estLieA5Traits(Fct,T1,S1,T2,S2,T3,S3,T4,S4,T5,S5,Ag),
aPourValeurTrait(T1,Esp,V1,CC), not aPourValeurTrait(T2,Esp,V2,CC),
aPourValeurTrait(T3,Esp,V3,CC), not aPourValeurTrait(T4,Esp,V4,CC),
aPourValeurTrait(T5,Esp,V5,CC).

aPourValeurFonctionCalculee(Fct,Esp,fct:agreg(Ag,V1,S1,V3,S3,V4,S4),60,CC)
:- estLieA5Traits(Fct,T1,S1,T2,S2,T3,S3,T4,S4,T5,S5,Ag),
aPourValeurTrait(T1,Esp,V1,CC), not aPourValeurTrait(T2,Esp,V2,CC),
aPourValeurTrait(T3,Esp,V3,CC), aPourValeurTrait(T4,Esp,V4,CC),
not aPourValeurTrait(T5,Esp,V5,CC).

aPourValeurFonctionCalculee(Fct,Esp,fct:agreg(Ag,V1,S1,V2,S2,V5,S5),60,CC)
:- estLieA5Traits(Fct,T1,S1,T2,S2,T3,S3,T4,S4,T5,S5,Ag),
aPourValeurTrait(T1,Esp,V1,CC), aPourValeurTrait(T2,Esp,V2,CC),
not aPourValeurTrait(T3,Esp,V3,CC), not aPourValeurTrait(T4,Esp,V4,CC),
aPourValeurTrait(T5,Esp,V5,CC).

aPourValeurFonctionCalculee(Fct,Esp,fct:agreg(Ag,V1,S1,V2,S2,V4,S4),60,CC)
:- estLieA5Traits(Fct,T1,S1,T2,S2,T3,S3,T4,S4,T5,S5,Ag),
aPourValeurTrait(T1,Esp,V1,CC), aPourValeurTrait(T2,Esp,V2,CC),
not aPourValeurTrait(T3,Esp,V3,CC),  aPourValeurTrait(T4,Esp,V4,CC),
not aPourValeurTrait(T5,Esp,V5,CC).

aPourValeurFonctionCalculee(Fct,Esp,fct:agreg(Ag,V1,S1,V2,S2,V3,S3),60,CC)
:- estLieA5Traits(Fct,T1,S1,T2,S2,T3,S3,T4,S4,T5,S5,Ag),
aPourValeurTrait(T1,Esp,V1,CC), aPourValeurTrait(T2,Esp,V2,CC),
aPourValeurTrait(T3,Esp,V3,CC), not aPourValeurTrait(T4,Esp,V4,CC),
not aPourValeurTrait(T5,Esp,V5,CC).


aPourValeurFonctionCalculee(Fct,Esp,fct:agreg(Ag,V4,S4,V5,S5),40,CC)
:- estLieA5Traits(Fct,T1,S1,T2,S2,T3,S3,T4,S4,T5,S5,Ag),
```

```
not aPourValeurTrait(T1,Esp,V1,CC), not aPourValeurTrait(T2,Esp,V2,CC),
not aPourValeurTrait(T3,Esp,V3,CC), aPourValeurTrait(T4,Esp,V4,CC),
aPourValeurTrait(T5,Esp,V5,CC).

aPourValeurFonctionCalculee(Fct,Esp,fct:agreg(Ag,V3,S3,V5,S5),40,CC)
:- estLieA5Traits(Fct,T1,S1,T2,S2,T3,S3,T4,S4,T5,S5,Ag),
not aPourValeurTrait(T1,Esp,V1,CC), not aPourValeurTrait(T2,Esp,V2,CC),
aPourValeurTrait(T3,Esp,V3,CC), not aPourValeurTrait(T4,Esp,V4,CC),
aPourValeurTrait(T5,Esp,V5,CC).

aPourValeurFonctionCalculee(Fct,Esp,fct:agreg(Ag,V3,S3,V4,S4),40,CC)
:- estLieA5Traits(Fct,T1,S1,T2,S2,T3,S3,T4,S4,T5,S5,Ag),
not aPourValeurTrait(T1,Esp,V1,CC), not aPourValeurTrait(T2,Esp,V2,CC),
aPourValeurTrait(T3,Esp,V3,CC),  aPourValeurTrait(T4,Esp,V4,CC),
not aPourValeurTrait(T5,Esp,V5,CC).

aPourValeurFonctionCalculee(Fct,Esp,fct:agreg(Ag,V2,S2,V5,S5),40,CC)
:- estLieA5Traits(Fct,T1,S1,T2,S2,T3,S3,T4,S4,T5,S5,Ag),
not aPourValeurTrait(T1,Esp,V1,CC), aPourValeurTrait(T2,Esp,V2,CC),
not aPourValeurTrait(T3,Esp,V3,CC), not aPourValeurTrait(T4,Esp,V4,CC),
aPourValeurTrait(T5,Esp,V5,CC).

aPourValeurFonctionCalculee(Fct,Esp,fct:agreg(Ag,V2,S2,V4,S4),40,CC)
:- estLieA5Traits(Fct,T1,S1,T2,S2,T3,S3,T4,S4,T5,S5,Ag),
not aPourValeurTrait(T1,Esp,V1,CC), aPourValeurTrait(T2,Esp,V2,CC),
not aPourValeurTrait(T3,Esp,V3,CC),  aPourValeurTrait(T4,Esp,V4,CC),
not aPourValeurTrait(T5,Esp,V5,CC).

aPourValeurFonctionCalculee(Fct,Esp,fct:agreg(Ag,V2,S2,V3,S3),40,CC)
:- estLieA5Traits(Fct,T1,S1,T2,S2,T3,S3,T4,S4,T5,S5,Ag),
not aPourValeurTrait(T1,Esp,V1,CC), aPourValeurTrait(T2,Esp,V2,CC),
aPourValeurTrait(T3,Esp,V3,CC), not aPourValeurTrait(T4,Esp,V4,CC),
not aPourValeurTrait(T5,Esp,V5,CC).

aPourValeurFonctionCalculee(Fct,Esp,fct:agreg(Ag,V1,S1,V5,S5),40,CC)
:- estLieA5Traits(Fct,T1,S1,T2,S2,T3,S3,T4,S4,T5,S5,Ag),
aPourValeurTrait(T1,Esp,V1,CC), not aPourValeurTrait(T2,Esp,V2,CC),
```

```
not aPourValeurTrait(T3,Esp,V3,CC), not  aPourValeurTrait(T4,Esp,V4,CC),
aPourValeurTrait(T5,Esp,V5,CC).

aPourValeurFonctionCalculee(Fct,Esp,fct:agreg(Ag,V1,S1,V4,S4),40,CC)
:- estLieA5Traits(Fct,T1,S1,T2,S2,T3,S3,T4,S4,T5,S5,Ag),
aPourValeurTrait(T1,Esp,V1,CC), not aPourValeurTrait(T2,Esp,V2,CC),
not aPourValeurTrait(T3,Esp,V3,CC),  aPourValeurTrait(T4,Esp,V4,CC),
not aPourValeurTrait(T5,Esp,V5,CC).

aPourValeurFonctionCalculee(Fct,Esp,fct:agreg(Ag,V1,S1,V3,S3),40,CC)
:- estLieA5Traits(Fct,T1,S1,T2,S2,T3,S3,T4,S4,T5,S5,Ag),
aPourValeurTrait(T1,Esp,V1,CC), not aPourValeurTrait(T2,Esp,V2,CC),
aPourValeurTrait(T3,Esp,V3,CC), not aPourValeurTrait(T4,Esp,V4,CC),
not aPourValeurTrait(T5,Esp,V5,CC).

aPourValeurFonctionCalculee(Fct,Esp,fct:agreg(Ag,V1,S1,V2,S2),40,CC)
:- estLieA5Traits(Fct,T1,S1,T2,S2,T3,S3,T4,S4,T5,S5,Ag),
aPourValeurTrait(T1,Esp,V1,CC), aPourValeurTrait(T2,Esp,V2,CC),
not aPourValeurTrait(T3,Esp,V3,CC), not aPourValeurTrait(T4,Esp,V4,CC),
not aPourValeurTrait(T5,Esp,V5,CC).

aPourValeurFonctionCalculee(Fct,Esp,fct:une_valeur(V5,S5),20,CC)
:- estLieA5Traits(Fct,T1,S1,T2,S2,T3,S3,T4,S4,T5,S5,Ag),
not  aPourValeurTrait(T1,Esp,V1,CC), not aPourValeurTrait(T2,Esp,V2,CC),
not aPourValeurTrait(T3,Esp,V3,CC), not  aPourValeurTrait(T4,Esp,V4,CC),
aPourValeurTrait(T5,Esp,V5,CC).

aPourValeurFonctionCalculee(Fct,Esp,fct:une_valeur(V4,S4),20,CC)
:- estLieA5Traits(Fct,T1,S1,T2,S2,T3,S3,T4,S4,T5,S5,Ag),
not  aPourValeurTrait(T1,Esp,V1,CC), not aPourValeurTrait(T2,Esp,V2,CC),
not aPourValeurTrait(T3,Esp,V3,CC),   aPourValeurTrait(T4,Esp,V4,CC),
not aPourValeurTrait(T5,Esp,V5,CC).

aPourValeurFonctionCalculee(Fct,Esp,fct:une_valeur(V3,S3),20,CC)
:- estLieA5Traits(Fct,T1,S1,T2,S2,T3,S3,T4,S4,T5,S5,Ag),
not  aPourValeurTrait(T1,Esp,V1,CC), not aPourValeurTrait(T2,Esp,V2,CC),
aPourValeurTrait(T3,Esp,V3,CC), not  aPourValeurTrait(T4,Esp,V4,CC),
```

```
not aPourValeurTrait(T5,Esp,V5,CC).

aPourValeurFonctionCalculee(Fct,Esp,fct:une_valeur(V2,S2),20,CC)
:- estLieA5Traits(Fct,T1,S1,T2,S2,T3,S3,T4,S4,T5,S5,Ag),
not  aPourValeurTrait(T1,Esp,V1,CC), aPourValeurTrait(T2,Esp,V2,CC),
not aPourValeurTrait(T3,Esp,V3,CC), not  aPourValeurTrait(T4,Esp,V4,CC),
not aPourValeurTrait(T5,Esp,V5,CC).

aPourValeurFonctionCalculee(Fct,Esp,fct:une_valeur(V1,S1),20,CC)
:- estLieA5Traits(Fct,T1,S1,T2,S2,T3,S3,T4,S4,T5,S5,Ag),
aPourValeurTrait(T1,Esp,V1,CC), not aPourValeurTrait(T2,Esp,V2,CC),
not aPourValeurTrait(T3,Esp,V3,CC), not  aPourValeurTrait(T4,Esp,V4,CC),
not aPourValeurTrait(T5,Esp,V5,CC).

% Prise en compte des retours utilisateurs - Niveau fonctions
aPourValeurFonction(Fct,Esp,V,Fiab,CC):-
aPourValeurFonctionCalculee(Fct,Esp,V,Fiab,CC),
not aPourValeurFonctionUtilisateur(Fct,Esp,X,Y,CC).

aPourValeurFonction(Fct,Esp,V,Fiab,CC):-
aPourValeurFonctionUtilisateur(Fct,Esp,V,Fiab,CC).

% Évaluation d'un service par rapport aux valeurs des fonctions
% liées pour une espèce donnée
%%%% Service lié à 2 fonctions
aPourValeurServiceCalculee(S,Esp,fct:agreg(Ag,V1,S1,V2,S2),
fct:agreg("moy",Fiab1,NbTrait1,Fiab2,NbTrait2),CC)
:- estLieA2Fonctions(S,Fct1,S1,Fct2,S2,Ag),
aPourValeurFonction(Fct1,Esp,V1,Fiab1,CC), nbrTrait(Fct1,NbTrait1),
aPourValeurFonction(Fct2,Esp,V2,Fiab2,CC), nbrTrait(Fct2,NbTrait2).

aPourValeurServiceCalculee(S,Esp,fct:une_valeur(V1,S1),
fct:agreg("moy",Fiab1,NbTrait1,0,NbTrait2),CC)
:- estLieA2Fonctions(S,Fct1,S1,Fct2,S2,Ag),
aPourValeurFonction(Fct1,Esp,V1,Fiab1,CC), nbrTrait(Fct1,NbTrait1),
not aPourValeurFonction(Fct2,Esp,X,Y,CC), nbrTrait(Fct2,NbTrait2).
```

```
aPourValeurServiceCalculee(S,Esp,fct:une_valeur(V2,S2),
fct:agreg("moy",0,NbTrait1,Fiab2,NbTrait2),CC)
:- estLieA2Fonctions(S,Fct1,S1,Fct2,S2,Ag),
not aPourValeurFonction(Fct1,Esp,V1,Fiab1,CC), nbrTrait(Fct1,NbTrait1),
aPourValeurFonction(Fct2,Esp,V2,Fiab2,CC), nbrTrait(Fct2,NbTrait2).

%%%% Service lié à 3 fonctions
aPourValeurServiceCalculee(S,Esp,fct:agreg(Ag,V1,S1,V2,S2,V3,S3),
fct:agreg("moy",Fiab1,NbTrait1,Fiab2,NbTrait2,P3,NbTrait3),CC)
:- estLieAFonction3(S,Fct1,S1,Fct2,S2,Fct3,S3,Ag),
aPourValeurFonction(Fct1,Esp,V1,Fiab1,CC), nbrTrait(Fct1,NbTrait1),
aPourValeurFonction(Fct2,Esp,V2,Fiab2,CC), nbrTrait(Fct2,NbTrait2),
aPourValeurFonction(Fct3,Esp,V3,P3,CC), nbrTrait(Fct3,NbTrait3).

aPourValeurServiceCalculee(S,Esp,fct:agreg(Ag,V1,S1,V2,S2),
fct:agreg("moy",Fiab1,NbTrait1,Fiab2,NbTrait2,0,NbTrait3),CC)
:- estLieAFonction3(S,Fct1,S1,Fct2,S2,Fct3,S3,Ag),
aPourValeurFonction(Fct1,Esp,V1,Fiab1,CC), nbrTrait(Fct1,NbTrait1),
aPourValeurFonction(Fct2,Esp,V2,Fiab2,CC), nbrTrait(Fct2,NbTrait2),
not aPourValeurFonction(Fct3,Esp,X,Y,CC), nbrTrait(Fct3,NbTrait3).

aPourValeurServiceCalculee(S,Esp,fct:agreg(Ag,V1,S1,V3,S3),
fct:agreg("moy",Fiab1,NbTrait1,0,NbTrait2,Fiab3,NbTrait3),CC)
:- estLieAFonction3(S,Fct1,S1,Fct2,S2,Fct3,S3,Ag),
aPourValeurFonction(Fct1,Esp,V1,Fiab1,CC), nbrTrait(Fct1,NbTrait1),
not aPourValeurFonction(Fct2,Esp,V2,Fiab2,CC), nbrTrait(Fct2,NbTrait2),
aPourValeurFonction(Fct3,Esp,V3,Fiab3,CC), nbrTrait(Fct3,NbTrait3).

aPourValeurServiceCalculee(S,Esp,fct:agreg(Ag,V2,S2,V3,S3),
fct:agreg("moy",0,NbTrait1,Fiab2,NbTrait2,Fiab3,NbTrait3),CC)
:- estLieAFonction3(S,Fct1,S1,Fct2,S2,Fct3,S3,Ag),
not aPourValeurFonction(Fct1,Esp,V1,Fiab1,CC), nbrTrait(Fct1,NbTrait1),
aPourValeurFonction(Fct2,Esp,V2,Fiab2,CC), nbrTrait(Fct2,NbTrait2),
aPourValeurFonction(Fct3,Esp,V3,Fiab3,CC), nbrTrait(Fct3,NbTrait3).


aPourValeurServiceCalculee(S,Esp,fct:une_valeur(V3,S3),
```

```
fct:agreg("moy",0,NbTrait1,0,NbTrait2,Fiab3,NbTrait3),CC)
:- estLieAFonction3(S,Fct1,S1,Fct2,S2,Fct3,S3,Ag),
not aPourValeurFonction(Fct1,Esp,V1,Fiab1,CC), nbrTrait(Fct1,NbTrait1),
not aPourValeurFonction(Fct2,Esp,V2,Fiab2,CC), nbrTrait(Fct2,NbTrait2),
aPourValeurFonction(Fct3,Esp,V3,Fiab3,CC), nbrTrait(Fct3,NbTrait3).

aPourValeurServiceCalculee(S,Esp,fct:une_valeur(V2,S2),
fct:agreg("moy",0,NbTrait1,Fiab2,NbTrait2,0,NbTrait3),CC)
:- estLieAFonction3(S,Fct1,S1,Fct2,S2,Fct3,S3,Ag),
not aPourValeurFonction(Fct1,Esp,V1,Fiab1,CC), nbrTrait(Fct1,NbTrait1),
aPourValeurFonction(Fct2,Esp,V2,Fiab2,CC), nbrTrait(Fct2,NbTrait2),
not aPourValeurFonction(Fct3,Esp,V3,Fiab3,CC), nbrTrait(Fct3,NbTrait3).

aPourValeurServiceCalculee(S,Esp,fct:une_valeur(V1,S1),
fct:agreg("moy",Fiab1,NbTrait1,0,NbTrait2,0,NbTrait3),CC)
:- estLieAFonction3(S,Fct1,S1,Fct2,S2,Fct3,S3,Ag),
aPourValeurFonction(Fct1,Esp,V1,Fiab1,CC), nbrTrait(Fct1,NbTrait1),
not aPourValeurFonction(Fct2,Esp,V2,Fiab2,CC), nbrTrait(Fct2,NbTrait2),
not aPourValeurFonction(Fct3,Esp,V3,Fiab3,CC), nbrTrait(Fct3,NbTrait3).


%%%% Service lié à 4 fonctions
aPourValeurServiceCalculee(S,Esp,fct:agreg(Ag,V1,S1,V2,S2,V3,S3,V4,S4),
fct:agreg("moy",Fiab1,NbTrait1,Fiab2,NbTrait2,Fiab3,NbTrait3,Fiab4,NbTrait4),CC)
:- estLieAFonction4(S,Fct1,S1,Fct2,S2,Fct3,S3,Fct4,S4,Ag),
aPourValeurFonction(Fct1,Esp,V1,Fiab1,CC), nbrTrait(Fct1,NbTrait1),
aPourValeurFonction(Fct2,Esp,V2,Fiab2,CC), nbrTrait(Fct2,NbTrait2),
aPourValeurFonction(Fct3,Esp,V3,Fiab3,CC),nbrTrait(Fct3,NbTrait3),
aPourValeurFonction(Fct4,Esp,V4,Fiab4,CC), nbrTrait(Fct4,NbTrait4).

aPourValeurServiceCalculee(S,Esp,fct:agreg(Ag,V1,S1,V2,S2,V3,S3),
fct:agreg("moy",Fiab1,NbTrait1,Fiab2,NbTrait2,Fiab3,NbTrait3,0,NbTrait4),CC)
:- estLieAFonction4(S,Fct1,S1,Fct2,S2,Fct3,S3,Fct4,S4,Ag),
aPourValeurFonction(Fct1,Esp,V1,Fiab1,CC), nbrTrait(Fct1,NbTrait1),
aPourValeurFonction(Fct2,Esp,V2,Fiab2,CC), nbrTrait(Fct2,NbTrait2),
aPourValeurFonction(Fct3,Esp,V3,Fiab3,CC), nbrTrait(Fct3,NbTrait3),
nbrTrait(Fct4,NbTrait4), not aPourValeurFonction(Fct4,Esp,X,Y,CC).
```

```
aPourValeurServiceCalculee(S,Esp,fct:agreg(Ag,V1,S1,V2,S2,V4,S4),
fct:agreg("moy",Fiab1,NbTrait1,Fiab2,NbTrait2,0,NbTrait3,Fiab4,NbTrait4),CC)
:- estLieAFonction4(S,Fct1,S1,Fct2,S2,Fct3,S3,Fct4,S4,Ag),
aPourValeurFonction(Fct1,Esp,V1,Fiab1,CC), nbrTrait(Fct1,NbTrait1),
aPourValeurFonction(Fct2,Esp,V2,Fiab2,CC), nbrTrait(Fct2,NbTrait2),
not aPourValeurFonction(Fct3,Esp,V3,Fiab3,CC), nbrTrait(Fct3,NbTrait3),
nbrTrait(Fct4,NbTrait4), aPourValeurFonction(Fct4,Esp,V4,Fiab4,CC).

aPourValeurServiceCalculee(S,Esp,fct:agreg(Ag,V1,S1,V3,S3,V4,S4),
fct:agreg("moy",Fiab1,NbTrait1,0,NbTrait2,Fiab3,NbTrait3,Fiab4,NbTrait4),CC)
:- estLieAFonction4(S,Fct1,S1,Fct2,S2,Fct3,S3,Fct4,S4,Ag),
aPourValeurFonction(Fct1,Esp,V1,Fiab1,CC), nbrTrait(Fct1,NbTrait1),
not aPourValeurFonction(Fct2,Esp,V2,Fiab2,CC), nbrTrait(Fct2,NbTrait2),
aPourValeurFonction(Fct3,Esp,V3,Fiab3,CC), nbrTrait(Fct3,NbTrait3),
nbrTrait(Fct4,NbTrait4), aPourValeurFonction(Fct4,Esp,V4,Fiab4,CC).

aPourValeurServiceCalculee(S,Esp,fct:agreg(Ag,V2,S2,V3,S3,V4,S4),
fct:agreg("moy",0,NbTrait1,Fiab2,NbTrait2,Fiab3,NbTrait3,Fiab4,NbTrait4),CC)
:- estLieAFonction4(S,Fct1,S1,Fct2,S2,Fct3,S3,Fct4,S4,Ag),
not aPourValeurFonction(Fct1,Esp,V1,Fiab1,CC), nbrTrait(Fct1,NbTrait1),
aPourValeurFonction(Fct2,Esp,V2,Fiab2,CC), nbrTrait(Fct2,NbTrait2),
aPourValeurFonction(Fct3,Esp,V3,Fiab3,CC), nbrTrait(Fct3,NbTrait3),
nbrTrait(Fct4,NbTrait4), aPourValeurFonction(Fct4,Esp,V4,Fiab4,CC).

aPourValeurServiceCalculee(S,Esp,fct:agreg(Ag,V3,S3,V4,S4),
fct:agreg("moy",0,NbTrait1,0,NbTrait2,Fiab3,NbTrait3,Fiab4,NbTrait4),CC)
:- estLieAFonction4(S,Fct1,S1,Fct2,S2,Fct3,S3,Fct4,S4,Ag),
not aPourValeurFonction(Fct1,Esp,V1,Fiab1,CC), nbrTrait(Fct1,NbTrait1),
not aPourValeurFonction(Fct2,Esp,V2,Fiab2,CC), nbrTrait(Fct2,NbTrait2),
aPourValeurFonction(Fct3,Esp,V3,Fiab3,CC), nbrTrait(Fct3,NbTrait3),
nbrTrait(Fct4,NbTrait4), aPourValeurFonction(Fct4,Esp,V4,Fiab4,CC).

aPourValeurServiceCalculee(S,Esp,fct:agreg(Ag,V2,S2,V4,S4),
fct:agreg("moy",0,NbTrait1,Fiab2,NbTrait2,0,NbTrait3,Fiab4,NbTrait4),CC)
:- estLieAFonction4(S,Fct1,S1,Fct2,S2,Fct3,S3,Fct4,S4,Ag),
not aPourValeurFonction(Fct1,Esp,V1,Fiab1,CC), nbrTrait(Fct1,NbTrait1),
```

```
aPourValeurFonction(Fct2,Esp,V2,Fiab2,CC), nbrTrait(Fct2,NbTrait2),
not aPourValeurFonction(Fct3,Esp,V3,Fiab3,CC), nbrTrait(Fct3,NbTrait3),
nbrTrait(Fct4,NbTrait4), aPourValeurFonction(Fct4,Esp,V4,Fiab4,CC).


aPourValeurServiceCalculee(S,Esp,fct:agreg(Ag,V2,S2,V3,S3),
fct:agreg("moy",0,NbTrait1,Fiab2,NbTrait2,Fiab3,NbTrait3,0,NbTrait4),CC)
:- estLieAFonction4(S,Fct1,S1,Fct2,S2,Fct3,S3,Fct4,S4,Ag),
not aPourValeurFonction(Fct1,Esp,V1,Fiab1,CC), nbrTrait(Fct1,NbTrait1),
aPourValeurFonction(Fct2,Esp,V2,Fiab2,CC), nbrTrait(Fct2,NbTrait2),
aPourValeurFonction(Fct3,Esp,V3,Fiab3,CC), nbrTrait(Fct3,NbTrait3),
nbrTrait(Fct4,NbTrait4), not aPourValeurFonction(Fct4,Esp,V4,Fiab4,CC).

aPourValeurServiceCalculee(S,Esp,fct:agreg(Ag,V1,S1,V4,S4),
fct:agreg("moy",Fiab1,NbTrait1,0,NbTrait2,0,NbTrait3,Fiab4,NbTrait4),CC)
:- estLieAFonction4(S,Fct1,S1,Fct2,S2,Fct3,S3,Fct4,S4,Ag),
aPourValeurFonction(Fct1,Esp,V1,Fiab1,CC), nbrTrait(Fct1,NbTrait1),
not aPourValeurFonction(Fct2,Esp,V2,Fiab2,CC), nbrTrait(Fct2,NbTrait2),
not aPourValeurFonction(Fct3,Esp,V3,Fiab3,CC), nbrTrait(Fct3,NbTrait3),
nbrTrait(Fct4,NbTrait4), aPourValeurFonction(Fct4,Esp,V4,Fiab4,CC).

aPourValeurServiceCalculee(S,Esp,fct:agreg(Ag,V1,S1,V3,S3),
fct:agreg("moy",Fiab1,NbTrait1,0,NbTrait2,Fiab3,NbTrait3,0,NbTrait4),CC)
:- estLieAFonction4(S,Fct1,S1,Fct2,S2,Fct3,S3,Fct4,S4,Ag),
aPourValeurFonction(Fct1,Esp,V1,Fiab1,CC), nbrTrait(Fct1,NbTrait1),
not aPourValeurFonction(Fct2,Esp,V2,Fiab2,CC), nbrTrait(Fct2,NbTrait2),
aPourValeurFonction(Fct3,Esp,V3,Fiab3,CC), nbrTrait(Fct3,NbTrait3),
nbrTrait(Fct4,NbTrait4), not aPourValeurFonction(Fct4,Esp,V4,Fiab4,CC).

aPourValeurServiceCalculee(S,Esp,fct:agreg(Ag,V1,S1,V2,S2),
fct:agreg("moy",Fiab1,NbTrait1,Fiab2,NbTrait2,0,NbTrait3,0,NbTrait4),CC)
:- estLieAFonction4(S,Fct1,S1,Fct2,S2,Fct3,S3,Fct4,S4,Ag),
aPourValeurFonction(Fct1,Esp,V1,Fiab1,CC), nbrTrait(Fct1,NbTrait1),
aPourValeurFonction(Fct2,Esp,V2,Fiab2,CC), nbrTrait(Fct2,NbTrait2),
not aPourValeurFonction(Fct3,Esp,V3,Fiab3,CC), nbrTrait(Fct3,NbTrait3),
nbrTrait(Fct4,NbTrait4), not aPourValeurFonction(Fct4,Esp,V4,Fiab4,CC).
```

```
aPourValeurServiceCalculee(S,Esp,fct:une_valeur(V1,S1),
fct:agreg("moy",Fiab1,NbTrait1,0,NbTrait2,0,NbTrait3,0,NbTrait4),CC)
:- estLieAFonction4(S,Fct1,S1,Fct2,S2,Fct3,S3,Fct4,S4,Ag),
aPourValeurFonction(Fct1,Esp,V1,Fiab1,CC), nbrTrait(Fct1,NbTrait1),
not aPourValeurFonction(Fct2,Esp,V2,Fiab2,CC), nbrTrait(Fct2,NbTrait2),
not aPourValeurFonction(Fct3,Esp,V3,Fiab3,CC), nbrTrait(Fct3,NbTrait3),
nbrTrait(Fct4,NbTrait4), not aPourValeurFonction(Fct4,Esp,V4,Fiab4,CC).

aPourValeurServiceCalculee(S,Esp,fct:une_valeur(V2,S2),
fct:agreg("moy",0,NbTrait1,Fiab2,NbTrait2,0,NbTrait3,0,NbTrait4),CC)
:- estLieAFonction4(S,Fct1,S1,Fct2,S2,Fct3,S3,Fct4,S4,Ag),
not aPourValeurFonction(Fct1,Esp,V1,Fiab1,CC), nbrTrait(Fct1,NbTrait1),
aPourValeurFonction(Fct2,Esp,V2,Fiab2,CC), nbrTrait(Fct2,NbTrait2),
not aPourValeurFonction(Fct3,Esp,V3,Fiab3,CC), nbrTrait(Fct3,NbTrait3),
nbrTrait(Fct4,NbTrait4), not aPourValeurFonction(Fct4,Esp,V4,Fiab4,CC).

aPourValeurServiceCalculee(S,Esp,fct:une_valeur(V3,S3),
fct:agreg("moy",0,NbTrait1,0,NbTrait2,Fiab3,NbTrait3,0,NbTrait4),CC)
:- estLieAFonction4(S,Fct1,S1,Fct2,S2,Fct3,S3,Fct4,S4,Ag),
not aPourValeurFonction(Fct1,Esp,V1,Fiab1,CC), nbrTrait(Fct1,NbTrait1),
not aPourValeurFonction(Fct2,Esp,V2,Fiab2,CC), nbrTrait(Fct2,NbTrait2),
aPourValeurFonction(Fct3,Esp,V3,Fiab3,CC), nbrTrait(Fct3,NbTrait3),
nbrTrait(Fct4,NbTrait4), not aPourValeurFonction(Fct4,Esp,V4,Fiab4,CC).

aPourValeurServiceCalculee(S,Esp,fct:une_valeur(V4,S4),
fct:agreg("moy",0,NbTrait1,0,NbTrait2,0,NbTrait3,Fiab4,NbTrait4),CC)
:- estLieAFonction4(S,Fct1,S1,Fct2,S2,Fct3,S3,Fct4,S4,Ag),
not aPourValeurFonction(Fct1,Esp,V1,Fiab1,CC), nbrTrait(Fct1,NbTrait1),
not aPourValeurFonction(Fct2,Esp,V2,Fiab2,CC), nbrTrait(Fct2,NbTrait2),
not aPourValeurFonction(Fct3,Esp,V3,Fiab3,CC), nbrTrait(Fct3,NbTrait3),
nbrTrait(Fct4,NbTrait4), aPourValeurFonction(Fct4,Esp,V4,Fiab4,CC).

% Prise en compte des retours utilisateurs (services)
aPourValeurService(S,Esp,V,Fiab,CC):-
aPourValeurServiceCalculee(S,Esp,V,Fiab,CC),
not aPourValeurServiceUtilisateur(S,Esp,X,Y,CC).
```

```
aPourValeurService(S,Esp,V,Fiab,CC):-
aPourValeurServiceUtilisateur(S,Esp,V,Fiab,CC).

aPourValeurService(S,Esp,V,Fiab,CC):-
aPourValeurServiceCalculee(S,Esp,V,Fiab,CC),
not aPourValeurServiceUtilisateur(S,Esp,X,Y,CC).

aPourValeurService(S,Esp,V,Fiab,CC):-
aPourValeurServiceUtilisateur(S,Esp,V,Fiab,CC).
```

ANNEXE C

Graphe de dépendance des prédicats intensionnels

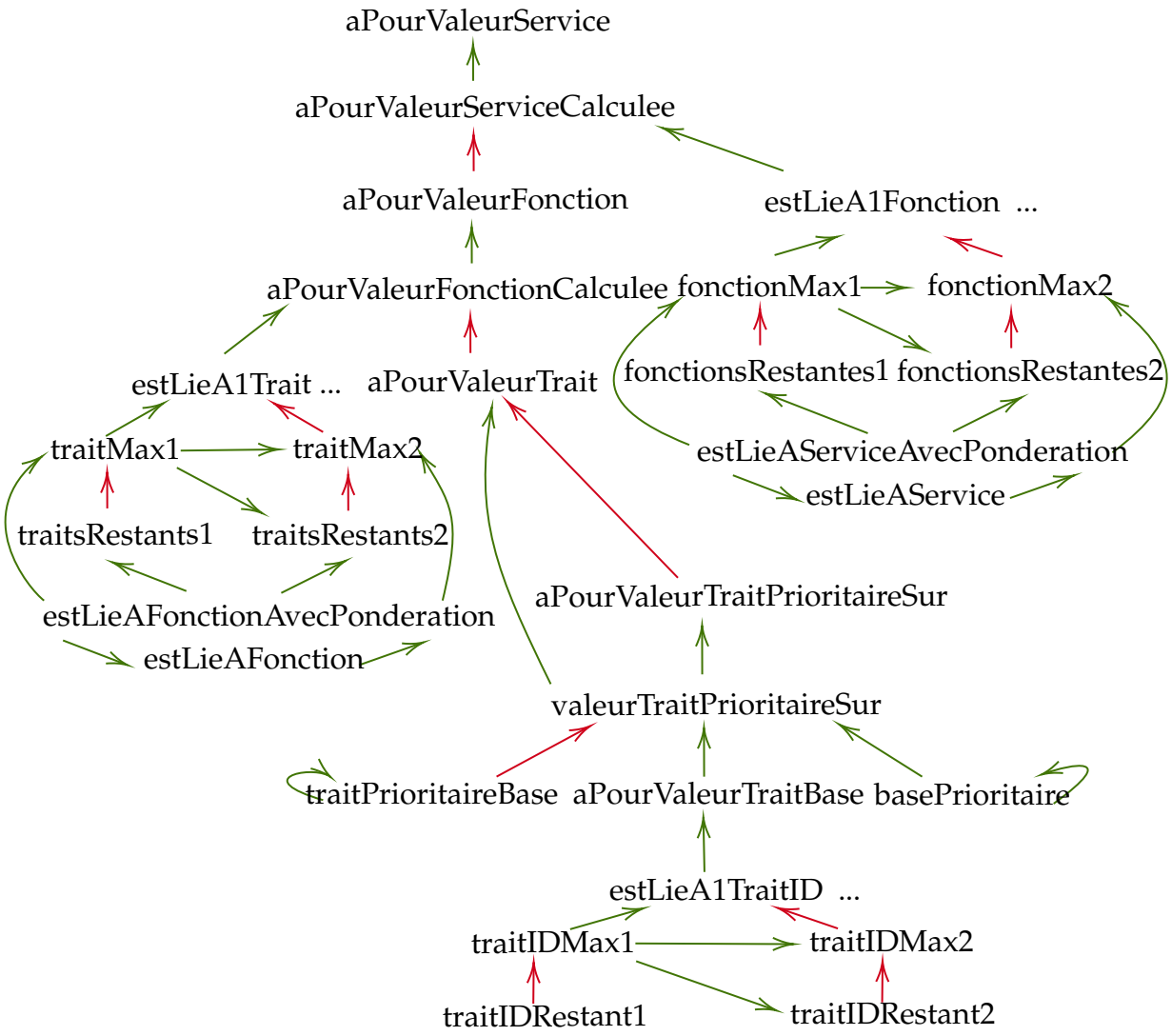


FIGURE C.1 – Graphe global de dépendance des prédicats intensionnels (sans la partie du graphe correspondant à l'ontologie).

Bibliographie

[Aba+19] Diego ABALOS, Jan Willem VAN GROENIGEN, Laurent PHILIPPOT, Ingrid M. LUBBERS et Gerlinde B. DE DEYN, « Plant trait-based approaches to improve nitrogen cycling in agroecosystems », English, in : *Journal of Applied Ecology* 56.*11* (nov. 2019), p. 2454-2466, ISSN : 0021-8901, DOI : 10.1111/1365-2664.13489.

[Abi+11] Serge ABITEBOUL, Ioana MANOLESCU, Philippe RIGAUX, Marie-Christine ROUSSET et Pierre SENELLART, *Web data management*, Cambridge University Press, 2011.

[AHV94] Serge ABITEBOUL, Richard HULL et Victor VIANU, *Foundations of Databases*, Addison Wesley, 1994.

[Baa+03] Franz BAADER, Diego CALVANESE, Deborah MCGUINNESS, Peter PATEL-SCHNEIDER, Daniele NARDI et al., *The description logic handbook : Theory, implementation and applications*, Cambridge university press, 2003.

[Bag+15] Jean-François BAGET, Michel LECLÈRE, Marie-Laure MUGNIER, Swan ROCHER et Clément SIPIETER, « Graal : A Toolkit for Query Answering with Existential Rules », in : *Proceedings of RuleML 2015, Berlin, Germany, August 2-5, 2015, Proceedings*, sous la dir. de Nick BASSILIADES, Georg GOTTLOB, Fariba SADRI, Adrian PASCHKE et Dumitru ROMAN, t. 9202, Lecture Notes in Computer Science, Springer, 2015, p. 328-344.

[Ben+12] Paolo BENINCASA, Roberta PACE, Giacomo TOSTI et Francesco TEI, « Early interspecific interference in the wheat/faba bean (Triticum aestivum/ Vicia faba ssp. minor) and rapeseed/squarrosum clover (Brassica napus var. oleifera/Trifolium squarrosum) intercrops », in : *Italian Journal of Agronomy* 7 (mai 2012), DOI : 10.4081/ija.2012.e24.

[Ble18] Jennifer BLESH, « Functional traits in cover crop mixtures : Biological nitrogen fixation and multifunctionality », en, in : *Journal of Applied Ecology* 55 (2018), p. 38-48, ISSN : 1365-2664, DOI : 10.1111/1365-2664.13011, URL : http://onlinelibrary.wiley.com/doi/10.1111/1365-2664.13011/abstract (visité le 04/12/2017).

[BLHL01] Tim BERNERS-LEE, James HENDLER et Ora LASSILA, « The Semantic Web », in : (2001).

[Boc14] Christian BOCKSTALLER, « Les enjeux de l'analyse multicritère », in : *Séminaire de lancement de la plateforme MEANS*, 2014.

[Bor96] Alexander BORGIDA, « On the Relative Expressiveness of Description Logics and Predicate Logics », in : *Artif. Intell.* 82.*1-2* (1996), p. 353-367, DOI : 10.1016/0004-3702(96)00004-5, URL : https://doi.org/10.1016/0004-3702(96)00004-5.

[Boy+13] Brad BOYLE, Nicole HOPKINS, Zhenyuan LU, Juan Antonio RAYGOZA GARAY, Dmitry MOZZHERIN, Tony REES, Naim MATASCI, Martha L NARRO, William H PIEL, Sheldon J MCKAY et al., « The taxonomic name resolution service : an online tool for automated standardization of plant names », in : *BMC bioinformatics* 14.*1* (2013), p. 1-15.

[Bra+08] Lars Olav BRANDSÆTER, Heidi HEGGEN, Hugh RILEY, Erling STUBHAUG et Trond M. HENRIKSEN, « Winter survival, biomass accumulation and N mineralization of winter annual and biennial legumes sown at various times of year in Northern Temperate Regions », in : *European Journal of Agronomy* 28.*3* (2008), p. 437-448, ISSN : 1161-0301, DOI : https://doi.org/10.1016/j.eja.2007.11.013, URL : https://www.sciencedirect.com/science/article/pii/S1161030107001219.

[Bur+20a] Maxime BURON, François GOASDOUÉ, Ioana MANOLESCU et Marie-Laure MUGNIER, « Obi-Wan : Ontology-Based RDF Integration of Heterogeneous Data », in : *VLDB 2020 - 46th International Conference on Very Large Data Bases*, Tokyo, Japan, août 2020, URL : https://hal.inria.fr/hal-02921434.

[Bur+20b] Maxime BURON, François GOASDOUÉ, Ioana MANOLESCU et Marie-Laure MUGNIER, « Ontology-Based RDF Integration of Heterogeneous Data », in : *Proceedings of EDBT 2020, Copenhagen, Denmark, March 30 - April 02, 2020*, 2020, p. 299-310.

[BWR00] Liz BAGGS, Christine WATSON et Bob REES, « The fate of nitrogen from incorporated cover crop and green manure residues », in : *Nutrient Cycling in Agroecosystems* 56 (fév. 2000), p. 153-163, DOI : 10.1023/A:1009825606341.

[Cal+05] Diego CALVANESE, Giuseppe DE GIACOMO, Domenico LEMBO, Maurizio LENZERINI et Riccardo ROSATI, « DL-Lite : Tractable description logics for ontologies », in : *AAAI*, t. 5, 2005, p. 602-607.

[Cal+11] Diego CALVANESE, Giuseppe DE GIACOMO, Domenico LEMBO, Maurizio LENZERINI, Antonella POGGI, Mariano RODRIGUEZ-MURO, Riccardo ROSATI, Marco RUZZI et Domenico Fabio SAVO, « The MASTRO system for ontology-based data access », in : *Semantic Web* 2.1 (2011), p. 43-53.

[Cal+15] Diego CALVANESE, Benjamin COGREL, Elem Guzel KALAYCI, Sarah KOMLA-EBRI, Roman KONTCHAKOV, Davide LANTI, Martin REZK, Mariano RODRIGUEZ-MURO et Guohui XIAO, « OBDA with the Ontop Framework. », in : *SEBD*, Citeseer, 2015, p. 296-303.

[Cal+17] Diego CALVANESE, Benjamin COGREL, Sarah KOMLA-EBRI, Roman KONTCHAKOV, Davide LANTI, Martin REZK, Mariano RODRIGUEZ-MURO et Guohui XIAO, « Ontop : Answering SPARQL queries over relational databases », in : *Semantic Web* 8.3 (2017), p. 471-487, DOI : 10.3233/SW-160217, URL : https://doi.org/10.3233/SW-160217.

[Cap+04] Fabio CAPORALI, Enio CAMPIGLIA, Mancinelli ROBERTO, R PAOLINI et Ital AGRON, « Maize performances as influenced by winter cover crop green manuring », in : *Ital J Agron* 8 (jan. 2004).

[CGT+89] Stefano CERI, Georg GOTTLOB, Letizia TANCA et al., « What you always wanted to know about Datalog(and never dared to ask) », in : *IEEE transactions on knowledge and data engineering* 1.1 (1989), p. 146-166.

[Civ+13] Cristina CIVILI et al., « MASTRO STUDIO : Managing Ontology-Based Data Access applications », in : *Proc. VLDB Endow.* 6.12 (2013), p. 1314-1317, DOI : 10.14778/2536274.2536304, URL : http://www.vldb.org/pvldb/vol6/p1314-poggi.pdf.

[CM09] Michel CHEIN et Marie-Laure MUGNIER, *Graph-based Knowledge Representation - Computational Foundations of Conceptual Graphs*, Advanced Information and Knowledge Processing, Springer, 2009, ISBN : 978-1-84800-285-2, DOI : 10.1007/978-1-84800-286-9, URL : https://doi.org/10.1007/978-1-84800-286-9.

[Dam+15] Gaëlle DAMOUR, Eric GARNIER, Marie Laure NAVAS, Marc DOREL et Jean-Michel RISÈDE, « Chapter Three - Using Functional Traits to Assess the Services Provided by Cover Plants : A Review of Potentialities in Banana Cropping Systems », in : sous la dir. de Donald L. SPARKS, t. 134, Advances in Agronomy, Academic Press, 2015, p. 81-133, DOI : https://doi.org/10.1016/bs.agron.2015.06.004, URL : https://www.sciencedirect.com/science/article/pii/S0065211315001133.

[Dec+94] A Morris DECKER, Andrew J CLARK, John J MEISINGER, F Ronald MULFORD et Marla S MCINTOSH, « Legume cover crop contributions to no-tillage corn production », in : *Agronomy journal* 86.1 (1994), p. 126-135.

[DHI12] AnHai DOAN, Alon HALEVY et Zachary IVES, *Principles of data integration*, Elsevier, 2012.

[DNG18] Gaëlle DAMOUR, Marie L. NAVAS et Eric GARNIER, « A revised trait-based framework for agroecosystems including decision rules », en, in : *Journal of Applied Ecology* 55.1 (jan. 2018), sous la dir. d'Adam MARTIN, p. 12-24, ISSN : 00218901, DOI : 10.1111/1365-2664.12986, URL : http://doi.wiley.com/10.1111/1365-2664.12986 (visité le 07/05/2018).

[Dur+15] Michel DURU et al., « How to implement biodiversity-based agriculture to enhance ecosystem services : a review », in : *Agronomy for Sustainable Development* 35 (oct. 2015), DOI : 10.1007/s13593-015-0306-1.

[EFB84] SA EBELHAR, WW FRYE et RL BLEVINS, « Nitrogen from Legume Cover Crops for No-Tillage Corn 1 », in : *Agronomy journal* 76.1 (1984), p. 51-55.

[Fra20] Lauchlan H FRASER, « TRY—A plant trait database of databases », in : *Global change biology* 26.1 (2020), p. 189-190.

[FW16] Jennifer FUNK et Amelia WOLF, « Testing the Trait-Based Community Framework : Do Functional Traits Predict Competitive Outcomes ? », in : *Ecology* 97 (juin 2016), DOI : 10.1002/ecy.1484.

[Gar+17] Eric GARNIER et al., « Towards a thesaurus of plant characteristics : an ecological contribution », in : *Journal of Ecology* 105 (fév. 2017), 298–309, DOI : 10.1111/1365-2745.12698.

[Gar+18] Léo GARCIA, Florian CELETTE, Christian GARY, Aude RIPOCHE, Hector VALDÉS-GÓMEZ et Aurélie METAY, « Management of service crops for the provision of ecosystem services in vineyards : A review », in : *Agriculture, Ecosystems & Environment* 251 (2018), p. 158-170, ISSN : 0167-8809, DOI : https://doi.org/10.1016/j.agee.2017.09.030, URL : https://www.sciencedirect.com/science/article/pii/S0167880917304309.

[Gar+19] Léo GARCIA, Gaëlle DAMOUR, Christian GARY, Stéphane FOLLAIN, Yves LE BISSONNAIS et Aurélie METAY, « Trait-based approach for agroecology : contribution of service crop root traits to explain soil aggregate stability in vineyards », in : *Plant and Soil* 435 (fév. 2019), DOI : 10.1007/s11104-018-3874-4.

[Gar+20] Léo GARCIA, Aurélie METAY, Elena KAZAKOU, Jonathan STORKEY, Christian GARY et Gaëlle DAMOUR, « Optimizing the choice of service crops in vineyards to achieve both runoff mitigation and water provisioning for grapevine : a trait-based approach », en, in : *Plant and Soil* (mai 2020), ISSN : 0032-079X, 1573-5036, DOI : 10.1007/s11104-020-04543-y, URL : http://link.springer.com/10.1007/s11104-020-04543-y (visité le 22/05/2020).

[Gau+19] Noémie GAUDIO, Abraham J ESCOBAR-GUTIÉRREZ, Pierre CASADEBAIG, Jochem B EVERS, Frédéric GÉRARD, Gaëtan LOUARN, Nathalie COLBACH, Sebastian MUNZ, Marie LAUNAY, Hélène MARROU et al., « Current knowledge and future research opportunities for modeling annual crop mixtures. A review », in : *Agronomy for Sustainable Development* 39.2 (2019), p. 1-20.

[GN11] Eric GARNIER et Marie-Laure NAVAS, « A trait-based approach to comparative functional plant ecology : Concepts, methods and applications for agroecology. A review », in : *Agronomy for Sustainable Development* 32 (avr. 2011), p. 365-399, DOI : 10.1007/s13593-011-0036-y.

[Got+12] Georg GOTTLOB, Giorgio ORSI, Andreas PIERIS et Mantas ŠIMKUS, « Datalog and its extensions for semantic web databases », in : *Reasoning Web International Summer School*, Springer, 2012, p. 54-77.

[Gru95] Thomas R GRUBER, « Toward principles for the design of ontologies used for knowledge sharing ? », in : *International journal of human-computer studies* 43.5-6 (1995), p. 907-928.

[Han+20] Mario HANISCH, Oliver SCHWEIGER, Anna CORD, Martin VOLK et Sonja KNAPP, « Plant functional traits shape multiple ecosystem services, their trade-offs and synergies in grasslands », in : *Journal of Applied Ecology* 57 (avr. 2020), p. 1535-1550, DOI : 10.1111/1365-2664.13644.

[HJJ22] Christian HUYGHE, Florence JACQUET et Julia JOUAN, « La recherche pour une agriculture sans pesticides : un cadre disruptif aujourd'hui pour construire les solutions de demain », in : Editions QUAE, jan. 2022.

[HKR09] Pascal HITZLER, Markus KROTZSCH et Sebastian RUDOLPH, *Foundations of semantic web technologies*, Chapman et Hall/CRC, 2009.

[Ive+17] CM IVERSEN, AS POWELL, ML MCCORMACK, CB BLACKWOOD, GT FRESCHET, J KATTGE, C ROUMET, DB STOVER, NA SOUDZILOVSKAIA et OJ VALVERDE-BARRANTES, *Fine-root ecology database (FRED) : A Global collection of root trait data with coincident site, vegetation, edaphic, and climatic data, version 1*, rapp. tech., ORNLTESSFA (Oak Ridge National Lab's Terrestrial Ecosystem Science ..., 2017.

[Jai+05] Pankaj JAISWAL et al., « Plant Ontology (PO) : a Controlled Vocabulary of Plant Structures and Growth Stages », in : *Comparative and functional genomics* 6 (nov. 2005), p. 388-97, DOI : 10.1002/cfg.496.

[Jon+18] Clément JONQUET et al., « AgroPortal : A vocabulary and ontology repository for agronomy », in : *Computers and Electronics in Agriculture* 144 (2018), p. 126-143, ISSN : 0168-1699, DOI : https://doi.org/10.1016/j.compag.2017.10.012, URL : https://www.sciencedirect.com/science/article/pii/S0168169916309541.

[Ka20] Jens KATTGE et AL., « TRY plant trait database—enhanced coverage and open access », in : *Global Change Biology* 26.1 (2020), p. 119-188, DOI : https://doi.org/10.1111/gcb.14904, eprint : https://onlinelibrary.wiley.com/doi/pdf/10.1111/gcb.14904, URL : https://onlinelibrary.wiley.com/doi/abs/10.1111/gcb.14904.

[Kal12] Jesse M KALWIJ, « Review of 'The Plant List, a working list of all plant species' », in : *Journal of Vegetation Science* 23.5 (2012), p. 998-1002.

[Kal+20] Elem Güzel KALAYCI, Irlán GRANGEL-GONZÁLEZ, Felix LÖSCH, Guohui XIAO, Anees ul MEHDI, Evgeny KHARLAMOV et Diego CALVANESE, « Semantic Integration of Bosch Manufacturing Data Using Virtual Knowledge Graphs », in : *The Semantic Web - ISWC 2020 - 19th International Semantic Web Conference, Athens, Greece, November 2-6, 2020, Proceedings, Part II*, sous la dir. de Jeff Z. PAN, Valentina A. M. TAMMA, Claudia D'AMATO, Krzysztof JANOWICZ, Bo FU, Axel POLLERES, Oshani SENEVIRATNE et Lalana KAGAL, t. 12507, Lecture Notes in Computer Science, Springer, 2020, p. 464-481, DOI : 10.1007/978-3-030-62466-8_29, URL : https://doi.org/10.1007/978-3-030-62466-8_29.

[Kat+11] Jens KATTGE, Sandra DIAZ, Sandra LAVOREL, I Colin PRENTICE, Paul LEADLEY, Gerhard BÖNISCH, Eric GARNIER, Mark WESTOBY, Peter B REICH, Ian J WRIGHT et al., « TRY–a global database of plant traits », in : *Global change biology* 17.9 (2011), p. 2905-2935.

[Kat+20] Jens KATTGE et al., « TRY plant trait database – enhanced coverage and open access », English, in : *Global Change Biology* 26.1 (jan. 2020), p. 119-188, ISSN : 1354-1013, DOI : 10.1111/gcb.14904.

[KDK04] Ingolf KÜHN, Walter DURKA et Stefan KLOTZ, « BiolFlor : a new plant-trait database as a tool for plant invasion ecology », in : *Diversity and Distributions* 10.5/6 (2004), p. 363-365.

[Kem+22] Andrius Hansen KEMEZYS, Peter HARTVIG, Kaspar INGVORDSEN, Per Elmegaard ANDERSEN et Mie JENSEN, « Results of crop protection trials in minor crops in 2021 », English, in : *Applied Crop Protection 2021*, sous la dir. de Lise Nistrup JØRGENSEN, Thies Marten HEICK, Isaac Kwesi ABULEY, Andrius HANSEN et Helene Saltoft KRISTJANSEN, DCA rapport 204, Aarhus Universitet - DCA - Nationalt Center for Fødevarer og Jordbrug, mai 2022, p. 84-91, ISBN : 978-87-93998-85-8.

[Kha+17] Evgeny KHARLAMOV et al., « Ontology Based Data Access in Statoil », in : *J. Web Semant.* 44 (2017), p. 3-36, DOI : 10.1016/j.websem.2017.05.005, URL : https://doi.org/10.1016/j.websem.2017.05.005.

[Kha+18] Evgeny KHARLAMOV, Martin G. SKJÆVELAND, Dag HOVLAND, Theofilos MAILIS, Ernesto JIMÉNEZ-RUIZ, Guohui XIAO, Ahmet SOYLU, Ian HORROCKS et Arild WAALER, « Finding Data Should be Easier than Finding Oil », in : *IEEE International Conference on Big Data (IEEE BigData 2018), Seattle, WA, USA, December 10-13, 2018*, sous la dir. de Naoki ABE et al., IEEE, 2018, p. 1747-1756, DOI : 10.1109/BigData.2018.8622035, URL : https://doi.org/10.1109/BigData.2018.8622035.

[KJ02] Shiou KUO et Eric J. JELLUM, « Influence of Winter Cover Crop and Residue Management on Soil Nitrogen Availability and Corn », in : *Agronomy Journal* 94.3 (2002), p. 501-508, DOI : https://doi.org/10.2134/agronj2002.5010, eprint : https://acsess.onlinelibrary.wiley.com/doi/pdf/10.2134/agronj2002.5010, URL : https://acsess.onlinelibrary.wiley.com/doi/abs/10.2134/agronj2002.5010.

[Kne+03] IC KNEVEL, RM BEKKER, JP BAKKER et M KLEYER, « Life-history traits of the Northwest European flora : the LEDA database », in : *Journal of Vegetation Science* 14.4 (2003), p. 611-614.

[KS98] S. KUO et Upendra SAINJU, « Nitrogen mineralization and availability of mixed leguminous and non-leguminous cover crop residues in soil », in : *Biology and Fertility of Soils* 26 (jan. 1998), p. 346-353, DOI : 10.1007/s003740050387.

[KSJ96] S. KUO, Upendra SAINJU et E. JELLUM, « Winter cover cropping influence on nitrogen mineralization, presidedress soil nitrate test, and corn yields », in : *Biology and Fertility of Soils* 22 (juin 1996), p. 310-317, DOI : 10.1007/BF00334575.

[Len18] Maurizio LENZERINI, « Managing Data through the Lens of an Ontology », in : *AI Mag.* 39.2 (2018), p. 65-74, DOI : 10.1609/aimag.v39i2.2802, URL : https://doi.org/10.1609/aimag.v39i2.2802.

[Léo+20] Simon LÉO, G HEINRICH, Lafond DAVID, A GUÉRIN, P GUILLERMIN, Simon SYLVAINE, Neveu PASCAL, Christian GARY et Metral RAPHAËL, « SYGNAL, un outil de représentation des connaissances pour le diagnostic et la conception des systèmes viticoles. Application aux maladies du bois », in : *12ème Journée Scientifique Vigne–Vin*, 2020.

[Les+15a] Françoise LESCOURRET, Thierry DUTOIT, Freddy REY, François CÔTE, Marjolaine HAMELIN et Eric LICHTFOUSE, « Agroecological engineering », in : *Agron. Sustain. Dev.* 35 (2015), 1191–1198, DOI : 10.1007/s13593-015-0335-9.

[Les+15b] Françoise LESCOURRET et al., « A social–ecological approach to managing multiple agro-ecosystem services », in : *Current Opinion in Environmental Sustainability* 14 (2015), Open Issue, p. 68-75, ISSN : 1877-3435, DOI : https://doi.org/10.1016/j.cosust.2015.04.001, URL : https://www.sciencedirect.com/science/article/pii/S1877343515000391.

[Mal+09] Eric MALÉZIEUX, Yves CROZAT, Christian DUPRAZ, Marilyne LAURANS, David MAKOWSKI, Harry OZIER-LAFONTAINE, Bruno RAPIDEL, S de TOURDONNET et Muriel VALANTIN-MORISON, « Mixing plant species in cropping systems : concepts, tools and models : a review », in : *Sustainable agriculture* (2009), p. 329-353.

[Mat+13] Luca MATTEIS, Pierre-Yves CHIBON, Herlin ESPINOSA, Milko SKOFIC, Richard FINKERS, Richard BRUSKIEWICH, Glenn HYMAN et Elizabeth ARNAUD, « Crop Ontology : Vocabulary For Crop-related Concepts », in : t. 979, mai 2013.

[MB09] Alistair MILES et Sean BECHHOFER, « SKOS simple knowledge organization system reference », in : *W3C recommendation* 18 (2009), W3C.

[MBA22] Liane MIEDEMA BROWN et Madhur ANAND, « Plant functional traits as measures of ecosystem service provision », in : *Ecosphere* 13.2 (2022), e3930.

[Mey+22] Nicolas MEYER, Jacques-Eric BERGEZ, Eric JUSTES et Julie CONSTANTIN, « Influence of cover crop on water and nitrogen balances and cash crop yield in a temperate climate : A modelling approach using the STICS soil-crop model », in : *European Journal of Agronomy* 132 (2022), p. 126416.

[MI15] Adam R. MARTIN et Marney E. ISAAC, « Functional traits in agroecoogy : a blueprint for research », in : *Journal of Applied Ecology* 52 (2015), p. 1425-1435, ISSN : 00218901, DOI : 10.1111/1365-2664.12526.

[MI18] Adam R. MARTIN et Marney E. ISAAC, « Functional traits in agroecology : Advancing description and prediction in agroecosystems », en, in : *Journal of Applied Ecology* 55.1 (jan. 2018), p. 5-11, ISSN : 1365-2664, DOI : 10.1111/1365-2664.13039, URL : http://onlinelibrary.wiley.com/doi/10.1111/1365-2664.13039/abstract (visité le 13/12/2017).

[MT14] Marie-Laure MUGNIER et Michaël THOMAZO, « An introduction to ontology-based query answering with existential rules », in : *Reasoning Web International Summer School*, Springer, 2014, p. 245-278.

[Mug11] Marie-Laure MUGNIER, « Ontological query answering with existential rules », in : *International Conference on Web Reasoning and Rule Systems*, Springer, 2011, p. 2-23.

[OL+10] Harry OZIER-LAFONTAINE, Jean-Marc BLAZY, Mirza PUBLICOL et Cindy MELFORT, *SIMSERV - Expert system of selection assistance of service plants*, 2010, URL : https://hal.inrae.fr/hal-02820845.

[Pog+08] Antonella POGGI, Domenico LEMBO, Diego CALVANESE, Giuseppe De GIACOMO, Maurizio LENZERINI et Riccardo ROSATI, « Linking data to ontologies », in : *Journal on data semantics X*, Springer, 2008, p. 133-173.

[Puy] Philippe PUYDARRIEUX, « L'évaluation française des écosystèmes et des services d'écosystèmes (EFESE) », in : ().

[RK12] Sachit RAJBHANDARI et Johannes KEIZER, « The AGROVOC Concept Scheme – A Walkthrough », in : *Journal of Integrative Agriculture* 11.5 (2012), p. 694-699, ISSN : 2095-3119, DOI : https://doi.org/10.1016/S2095-3119(12)60058-6, URL : https://www.sciencedirect.com/science/article/pii/S2095311912600586.

[RW96] Noah N RANELLS et Michael G WAGGER, « Nitrogen release from grass and legume cover crop monocultures and bicultures », in : *Agronomy Journal* 88.5 (1996), p. 777-882.

[SAM14] Juan F. SEQUEDA, Marcelo ARENAS et Daniel P. MIRANKER, « OBDA : Query Rewriting or Materialization ? In Practice, Both ! », in : *ISWC*, 2014.

[She92] Carol SHENNAN, « Cover Crops, Nitrogen Cycling, and Soil Properties in Semi-irrigated Vegetable Production Systems », in : *HortScience HortSci* 27.7 (1992), p. 749 -754, DOI : 10.21273/HORTSCI.27.7.749.

[Sim+17] Sylvaine SIMON, Magalie LESUEUR-JANNOYER, Daniel PLÉNET, Pierre Éric LAURI et Fabrice LE BELLEC, « Methodology to design agroecological orchards : Learnings from on-station and on-farm experiences », in : *European Journal of Agronomy* 82 (2017), Farming systems analysis and design for sustainable intensification : new methods and assessments, p. 320-330, ISSN : 1161-0301, DOI : https://doi.org/10.1016/j.eja.2016.09.004, URL : https://www.sciencedirect.com/science/article/pii/S1161030116301654.

[SM13] Juan F SEQUEDA et Daniel P MIRANKER, « Ultrawrap : SPARQL execution on relational data », in : *Journal of Web Semantics* 22 (2013), p. 19-39.

[Sou+19] Vincent SOULIGNAC, François PINET, Eva LAMBERT, Laurence GUICHARD, Luce TROUCHE et Sophie AUBIN, « GECO, the French Web-based application for knowledge management in agroecology », in : *Computers and Electronics in Agriculture* 162 (2019), p. 1050-1056, ISSN : 0168-1699, DOI : https://doi.org/10.1016/j.compag.2017.10.028, URL : https://www.sciencedirect.com/science/article/pii/S0168169917300492.

[Sow84] John F. SOWA, *Conceptual Structures : Information Processing in Mind and Machine*, Addison-Wesley, 1984, ISBN : 0-201-14472-7.

[SS91] Lydia J. STIVERS et Carol SHENNAN, « Meeting the Nitrogen Needs of Processing Tomatoes through Winter Cover Cropping », in : *Journal of Production Agriculture* 4.3 (1991), p. 330-334, DOI : https://doi.org/10.2134/jpa1991.0330, eprint : https://acsess.onlinelibrary.wiley.com/doi/pdf/10.2134/jpa1991.0330, URL : https://acsess.onlinelibrary.wiley.com/doi/abs/10.2134/jpa1991.0330.

[TF12] Heather TURNER et David FIRTH, « Bradley-Terry Models in R : The BradleyTerry2 Package », in : *Journal of statistical software* 48 (mai 2012), DOI : 10.18637/jss.v048.i09.

[TP18] Çağatay TAVŞANOĞLU et Juli G PAUSAS, « A functional trait database for Mediterranean Basin plants », in : *Scientific Data* 5.1 (2018), p. 1-18.

[VDW+19] JUST VAN DER WOLF, LAURENCE JASSOGNE, GIL GRAM et PHILIPPE VAAST, « TURNING LOCAL KNOWLEDGE ON AGROFORESTRY INTO AN ONLINE DECISION-SUPPORT TOOL FOR TREE SELECTION IN SMALLHOLDERS' FARMS », in : *Experimental Agriculture* 55.*S1* (2019), 50–66, DOI : 10.1017/S001447971600017X.

[Vio+07] Cyrille VIOLLE, Marie-Laure NAVAS, Denis VILE, Elena KAZAKOU, Claire FORTUNEL, Irène HUMMEL et Eric GARNIER, « Let the concept of trait be functional ! Oikos », in : *Oikos* 116 (mai 2007), p. 882 -892, DOI : 10.1111/j.0030-1299.2007.15559.x.

[Vog+22] Iris VOGELER, Ingrid Kaag THOMSEN, Friedhelm TAUBE, Henrik Vestergaard POULSEN, Ralf LOGES et Elly Møller HANSEN, « Effect of winter cereal sowing time on yield and nitrogen leaching based on experiments and modelling », in : *Soil Use and Management* 38.1 (2022), p. 663-675, DOI : https://doi.org/10.1111/sum.12747, eprint : https://bsssjournals.onlinelibrary.wiley.com/doi/pdf/10.1111/sum.12747, URL : https://bsssjournals.onlinelibrary.wiley.com/doi/abs/10.1111/sum.12747.

[Wez+09] A. WEZEL, Stéphane BELLON, T. DORÉ, Charles FRANCIS, Dominique VALLOD et Christophe DAVID, « Agroecology as a Science, a Movement and a Practice », in : *http ://dx.doi.org/10.1051/agro/2009004* 29 (déc. 2009), p. 503-515, DOI : 10.1007/978-94-007-0394-0_3.

[Woo+15] Stephen A. WOOD, Daniel S. KARP, Fabrice DECLERCK, Claire KREMEN, Shahid NAEEM et Cheryl A. PALM, « Functional traits in agriculture : Agrobiodiversity and ecosystem services », in : *Trends in Ecology and Evolution* 30.9 (2015), p. 531-539, ISSN : 01695347, DOI : 10.1016/j.tree.2015.06.013, URL : http://dx.doi.org/10.1016/j.tree.2015.06.013.

[Xia+18] Guohui XIAO, Diego CALVANESE, Roman KONTCHAKOV, Domenico LEMBO, Antonella POGGI, Riccardo ROSATI et Michael ZAKHARYASCHEV, « Ontology-Based Data Access : A Survey », in : *IJCAI*, 2018.