Vers une formalisation en Coq de la provenance de données

Véronique Benzaken, Sarah Cohen-Boulakia, Évelyne Contejean, Chantal Keller, Rébecca Zucchini

Vers une formalisation en Coq de la provenance de données

Véronique Benzaken[1], Sarah Cohen-Boulakia[1], Évelyne Contejean[2], Chantal Keller[1], and Rébecca Zucchini[1,3]

[1] LRI, Université de Paris Sud, CNRS (UMR8623) - Université Paris Saclay
[2] LRI, CNRS (UMR8623), Université de Paris Sud - Université Paris Saclay
[3] École Normale Supérieure Paris-Saclay - Université Paris Saclay
prénom.nom@u-psud.fr

Résumé

Dans de multiples domaines scientifiques, de nombreuses données sont générées quotidiennement et doivent être analysées. Dans ces processus d'analyse, les données initiales sont combinées à d'autres jeux de données massifs. Pour garantir une interprétation correcte des résultats de ces analyses de données, il est crucial de pouvoir retracer la provenance des données produites à partir des données initiales. La communauté des bases de données a proposé un cadre formel unifiant de «semi-anneaux de provenance». L'objectif de cet article est de certifier a posteriori la correction d'une provenance. Pour ce faire, nous proposons une formalisation en Coq fondée sur le modèle de semi-anneaux de provenance pour des analyses de données exprimées en algèbre relationnelle. Nous introduisons ici notamment une preuve d'adéquation de cette provenance avec l'interprétation usuelle de l'algèbre relationnelle. Il s'agit d'une première étape vers la formalisation de langages centrés données avec des garanties fortes de provenance.

1 Introduction

L'explosion du volume de données générées par les multiples capteurs et équipements utilisés parfois quotidiennement – objets connectés, séquenceurs d'ADN, satellites sondeurs – pose de nouveaux défis dans le processus d'analyse de données. En effet, le passage de ces données brutes à l'acquisition de nouvelles connaissances nécessite une analyse fine de ces données massives, qu'il faut notamment comparer, croiser et combiner à d'autres jeux de données. De nombreuses données intermédiaires et finales sont alors générées à leur tour lors du processus d'analyse. Pour comprendre et pouvoir interpréter correctement le résultat d'une analyse, l'expert a besoin de connaître sa *provenance*, c'est-à-dire qu'il a besoin d'accéder à des informations relatives aux jeux de données qui ont été impliqués lors de l'analyse. Sans provenance, l'interprétation des résultats finaux peut être faussée et la reproductibilité de l'analyse ne plus être assurée.

De nombreuses formalisations de la provenance ont été proposées dans la communauté «bases de données» notamment autour du concept de «why provenance» [7] déterminant l'ensemble des combinaisons de lignes nécessaires pour qu'une ligne résultat existe. Déterminer la provenance d'un résultat de requête revient alors à propager les annotations associées aux données utilisées et combinées pour construire ce résultat. Selon les modèles, les relations sont annotées à des niveaux variés (depuis le niveau de la relation jusqu'à celui du n-uplet) et les annotations peuvent revêtir des formes diverses [11, 21, 14]. Green *et al.* ont introduit en 2007 un cadre unifiant pour ces modèles de provenance proposant de représenter la provenance sous la forme de semi-anneaux (*Provenance semirings*) [15, 17]. Néanmoins ce modèle n'a pas aujourd'hui de forme opérationnelle sûre : étant donnée une analyse de données effectuée, il n'existe pas d'implémentation capable de générer automatiquement la provenance d'un résultat et de

certifier que cette provenance est correcte, point crucial pour assurer la bonne interprétation des résultats et la reproductibilité des analyses.

L'objectif de cet article est de garantir formellement la provenance des données impliquées dans une analyse. Bien que les résultats soient indépendants de l'assistant de preuves choisi, nous présentons la formalisation telle que nous avons effectuée en Coq. Plus précisément, nous nous intéresserons ici aux analyses effectuant des transformations de données, fondées sur l'algèbre relationnelle (sélection de données, combinaison - ou jointure - entre les données de plusieurs tables, ...). Ces analyses capturent un large panel d'analyses de données qui sont opérationnelles lorsqu'elles sont exprimées en SQL. Dans ce contexte, la provenance se représente comme une combinaison des provenances (des données) initiales : les opérateurs de l'algèbre combinent ces annotations grâce aux opérations du semi-anneau, dans un procédé similaire à l'interprétation abstraite. La formalisation présentée ici est exécutable et donc se base sur le modèle fini usuel de SQL.

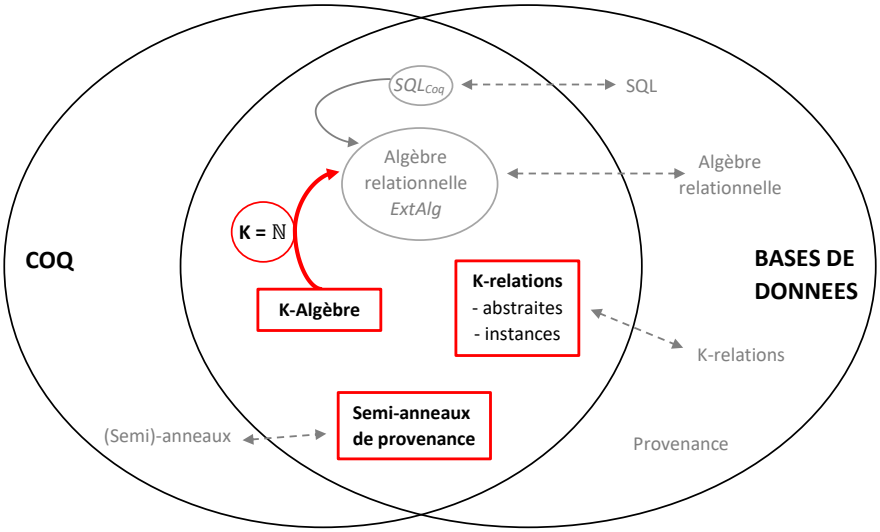

FIGURE 1 – Contributions

La Figure 1 introduit nos contributions et les place dans le paysage des bases de données et celui des bibliothèques de l'assistant Coq. Dans le domaine des bases de données, on retrouve la présence de l'algèbre relationnelle qui forme le noyau du langage SQL, les K-relations qui seront décrites plus précisément dans la Section 2 et qui constituent la base des semi-anneaux de provenance (relations annotées par des valeurs du semi-anneau K). Au niveau de Coq, on retrouve des bibliothèques dédiées aux semi-anneaux (indépendamment des K-relations) ainsi que SQLCoq et ExtAlg, des bibliothèques certifiées en Coq respectivement de SQL et de l'algèbre relationnelle [4, 3, 5].

Les contributions de l'article présent sont indiquées encadrées en trait gras (et en rouge). La première d'entre elles, présentée en Section 3, est la formalisation en Coq des semi-anneaux de provenance et de trois de leurs instances (entiers naturels, booléens, structures tropicales). La Section 4 présente notre formalisation en Coq des K-relations. La Section 5 présente notre théorème fondamental montrant l'adéquation des K-relations avec l'algèbre relationnelle classique, c'est-à-dire que la $\mathbb{N}$-Algèbre coïncide avec ExtAlg. Nous comparerons ensuite nos travaux avec l'état de l'art (Section 6) avant de conclure (Section 7).

2 Provenance et bases de données

2.1 Contexte

Une proposition pour être capable de suivre le cheminement des données au cours de transformations de données est d'utiliser des bases de données annotées. Chaque annotation contient des informations qui peuvent être de diverses natures : par exemple pour compter le nombre d'occurrences d'un n-uplet dans une relation donnée, pour informer de la présence ou de l'absence d'un n-uplet ou encore pour signaler le degré de sécurité nécessaire pour obtenir l'accès à un certain n-uplet. Les annotations permettent donc de tracer la provenance des différents n-uplets. Lors d'une requête, elles se propagent et subissent un certain nombre d'opérations.

	Cancer	Gène muté	Annot
1	Sein	BRCA1	a
2	Sein	BRCA2	b
3	Sang	RUNX1	c

Table 1 – NCBIGene

	Cancer	Gène muté	Annot
1	Sein	BRCA2	d
2	Foie	HULC	e

Table 2 – Refseq

	Cancer	Gène muté	Provenance
1	Sein	BRCA1	a
2	Sein	BRCA2	b $+_K$ d
3	Sang	RUNX1	c
4	Foie	HULC	e

Table 3 – NCBIGene ∪ Refseq

Cancer	Gène muté	Annot
Sein	BRCA1	a
Sein	BRCA2	b' = b+1
Sang	RUNX1	c

Table 4 – NCBIGene modifiée

	Cancer	Gène muté	Provenance
1	Sein	BRCA1	a
2	Sein	BRCA2	b' $+_K$ d
3	Sang	RUNX1	c
4	Foie	HULC	e

Table 5 – NCBIGene modifiée ∪ Refseq

Par exemple, prenons deux relations (Tables 1 et 2) représentant les patients atteints d'un type de cancer et présentant une certaine mutation dans leur génôme. Ces relations sont respectivement extraites des bases NBCIgene [19] et Refseq [22], toutes deux majeures pour les données de biologie moléculaire. Elles sont annotées par le nombre de patients atteints d'un certain cancer et ayant une certaine mutation. Nous souhaitons effectuer l'union des deux relations et observer les propagations des annotations. Les n-uplets 1 et 3 de la relation résultante (représentée dans la Table 3) proviennent de la relation NCBIGene uniquement et leurs annotations sont donc les mêmes que celles de la Table 1. De même, les annotations du n-uplet 4, qui provient de la relation Refseq, sont les mêmes que le n-uplet 2 de cette dernière. Enfin, le n-uplet 2 de la Table 3 provient des deux relations : les annotations sont donc une combinaison des annotations des deux relations, en l'occurrence l'addition (voir plus loin).

Les annotations permettent lors d'une modification d'un n-uplet de connaître l'impact de cette modification pour le résultat d'une requête. Dans la Table 1, si nous ajoutons un nouveau

patient qui a le cancer du sein et la mutation BRCA2, nous obtenons la TABLE 4 [1]. Lors de la requête effectuant l'union des deux tables, nous pouvons ainsi savoir que seul le deuxième n-uplet a été modifiée par l'ajout. En effet, le premier n-uplet de la TABLE 5 provient de la première ligne de la TABLE 1 et son annotation n'a pas été modifiée. Au contraire, le deuxième n-uplet de la TABLE 5 provient de la deuxième ligne de la TABLE 1 qui a été modifiée : b est donc remplacé par $b' = b + 1$.

Nous remarquons lors de l'union de deux n-uplets, l'utilisation de l'opérateur "+" au niveau des annotations. Dans le cas où le n-uplet est absent dans l'une des deux opérandes, le neutre "0" est utilisé, ce qui explique que l'annotation est inchangée. En outre, les opérateurs de l'algèbre relationnelle vérifiant des propriétés d'associativité, commutativité et de distributivité, il doit en aller de même pour les opérations sur les annotations : elles sont donc plongées *a minima* dans un monoïde commutatif.

2.2 Notations et définitions préliminaires

En pratique, pour capturer la richesse de l'algèbre relationnelle, les annotations sont plongées dans la structure algébrique unificatrice de semi-anneau commutatif [16].

Nous rappelons les définitions d'un monoïde, de la commutativité et d'un semi-anneau.

Définition 1 (Monoïde)**.** $(M, +, 0)$ est un monoïde si :

Loi interne $+$: $\forall x, y \in M^2, x + y \in M$

Associativité de $+$: $\forall x, y, z \in M^3, x + (y + z) = (x + y) + z$

0 **est neutre pour** $+$: $\forall x \in M, x + 0 = 0 + x = x$

Définition 2 (Commutativité)**.** Soit un ensemble M muni d'une loi interne $+$. La loi $+$ est commutative si : $\forall x, y \in M^2, x + y = y + x$.

Définition 3 (Semi-anneau)**.** $(K, 0, 1, +, \times)$ est un semi-anneau si :

1. $0 \neq 1$
2. $(K, +, 0)$ est un monoïde commutatif
3. $(K, \times, 1)$ est un monoïde
4. $\times$ est distributif par rapport à $+$: $\forall x, y, z \in K^3, x \times (y + z) = (x \times y) + (x \times z)$ et $\forall x, y, z \in K^3, (y + z) \times x = (y \times x) + (z \times y)$
5. 0 est absorbant pour le produit : $\forall x \in K, x \times 0 = 0 \times x = 0$

Lors d'une requête, les annotations subissent une combinaison d'opérations du semi-anneau commutatif K dans lequel elles sont plongées, combinaison reflétant les opérateurs de l'algèbre utilisés.

La notion usuelle de relation de l'algèbre relationnelle est un multi-ensemble de n-uplets, que l'on peut voir comme une fonction des n-uplets vers $\mathbb{N}$, indiquant le nombre d'occurrences de chaque n-uplet dans cette relation. L'idée de [16] est d'étendre les relations à des K-relations, plongeant ainsi à chaque n-uplet un élément de K.

Définition 4 (K-relations)**.** Soit $(K, 0, 1, +, *)$ un semi-anneau commutatif pour $*$ et r une relation. Une K-relation est une fonction de l'ensemble des n-uplets de r vers un élément de K.

1. Nous avons l'équivalence suivante sur les tables : une table ayant deux entrées avec le n-uplet t, l'une annoté avec a_1, l'autre avec a_2, est équivalente à une table identique à la première mais dans laquelle t a une seule entrée, annotée $a_1 +_K a_2$.

Nous noterons `annot(r,t)`, une fonction qui pour une relation r et un n-uplet t associe un élément de K. Nous nommons cet élément, l'annotation de t dans K.

Il est important de remarquer que nous nous plaçons dans le modèle usuel l'algèbre relationnelle, qui est fini. Les relations (tables) sont donc finies. En conséquence, dans le cadre des K-relations, cela revient à dire que la fonction `annot` doit être nulle presque partout, c'est-à-dire que l'ensemble des tuples t d'une relation r tels que `annot(r,t)`$\neq 0$ est fini.

Nous notons K-Algèbre, l'algèbre relationnelle étendue aux annotations plongées dans un semi-anneau commutatif K.

Nous présentons ici la syntaxe des requêtes utilisée et la sémantique de la K-Algèbre proposée par [16] avec quelques adaptations les rendant compatible avec les formalisations sur lesquelles nous nous basons dans la section suivante.

Nous utiliserons le terme *sorte* pour représenter l'ensemble des attributs d'une relation. Par exemple, la sorte de la relation de la Table 1 est {**Cancer**, **Gène muté**}.

Définition 5 (Syntaxe des requêtes)**.** La syntaxe des requêtes q est construite récursivement de la façon suivante :

$$q ::= \epsilon_{<>} \mid r \mid q \cup q \mid q \bowtie q \mid \pi(s, q) \mid \sigma(f, q)$$

où $\epsilon_{<>}$ est une relation de sorte vide et qui contient un n-uplet vide, r est une relation, l'opérateur $\cup$ correspond à l'union ensembliste, l'opérateur $\bowtie$ correspond à la jointure naturelle, l'opérateur π correspond à la projection, ici, sur un sous-ensemble s de la sorte de q, l'opérateur σ est la selection ici selon une formule f de la logique du premier ordre.

Nous notons $proj_s(t)$ la projection sur la sorte s du n-uplet t. Par exemple, la projection du n-uplet t={Sein ; BRCA1} sur la sorte contenant un seul attribut s={Cancer} nous donne le n-uplet $proj_s(t)$={Sein}.

Définition 6 (Sémantique)**.** Soit $(K, 0, 1, +, *)$ un semi-anneau commutatif. La sémantique $[\![q]\!]$ d'une requête q est une K-relation définie comme suit :

- $[\![\epsilon_{<>}]\!] = t \mapsto \begin{cases} 1 & \text{si } t = <> \\ 0 & \text{sinon} \end{cases}$
- $[\![r]\!] = t \mapsto$ `annot (r, t)`
- $[\![q_1 \cup q_2]\!] = t \mapsto [\![q_1]\!](t) + [\![q_2]\!](t)$
- $[\![q_1 \bowtie q_2]\!] = t \mapsto [\![q_1]\!](t_1) * [\![q_2]\!](t_2)$
  où les t_i correspondent aux projections de t sur la sorte des q_i associées.
- $[\![\pi(s, q)]\!] = t \mapsto \sum_{proj_s(t)=proj_s(t') \wedge [\![q]\!](t') \neq 0} [\![q]\!](t')$
- $[\![\sigma(f, q)]\!] = t \mapsto \begin{cases} [\![q]\!](t) & \text{si } eval(f, t) = \top \\ 0 & \text{sinon} \end{cases}$ où *eval* évalue la formule logique f pour le n-uplet t

Dans le cas de la projection π, la restriction $[\![q]\!](t') \neq 0$ permet d'avoir une somme finie. En effet, comme `annot` est nulle presque partout, sa généralisation à toute requête l'est aussi.

Pour s'assurer de la correction de la sémantique de la K-algèbre relationnelle, notre contribution principale est de formaliser en Coq son adéquation avec l'algèbre relationnelle classique dans le cas de $K = \mathbb{N}$. Pour cela, nous commençons par formaliser la structure mathématique régissant les annotations -les semi-anneaux commutatifs- et certaines instances de ces semi-anneaux commutatifs.

3 Semi-anneaux de provenance en Coq

Dans cette section, nous présentons la formalisation des semi-anneaux commutatifs en Coq et nous vérifions plusieurs instances de semi-anneaux utilisées pour la provenance : les entiers naturels, les booléens et les entiers tropicaux.

3.1 Formalisation des semi-anneaux

La formalisation s'inspire et se place dans la continuation de la bibliothèque Coccinelle [10], sur laquelle est basée l'algèbre relationnelle sans annotations que nous utilisons [4]. Celle-ci définit des structures algébriques sous forme de `Record` où chaque `Record` est paramétré par le type des éléments de la structure et un ordre total sur ce type. Par exemple, notre structure de base, le monoïde commutatif, est représenté par le type Coq ci-dessous.

```
Record CM (M : Type) (zero : M) (plus : M → M → M) (TM : total_order M) : Type := ...
```

Dans cette définition, le paramètre `TM : total_order M` munit `M` d'un ordre total, avec une fonction `compare : M → M → comparison`, la relation d'équivalence `eq : M → M → Prop` associée, et les axiomes usuels d'une relation d'ordre. Il est usuel d'utiliser en Coq une équalité décidable car nous nous plaçons dans le cadre de la logique intuitionniste. Nous supposons une propriété un peu plus forte (avoir un ordre total) mais en pratique, cette propriété est toujours vérifiée. L'ordre total permet de faciliter la formalisation en Coq.

La définition du `Record` énonce les axiomes du monoïde commutatif :

```
Record CM (M : Type) (zero : M) (plus : M → M → M) (TM : total_order M) : Type :=
mk_CM
  {
    plus_assocMC : ∀ a1 a2 a3, eq (plus a1 (plus a2 a3)) (plus (plus a1 a2) a3); (*+ associatif*)
    plus_zero_leftMC : ∀ a, eq (plus zero a) a; (*0 neutre de + a gauche*)
    plus_commMC : ∀ a1 a2, eq (plus a1 a2) (plus a2 a1);(*+ commutatif*)
    plus_compat_eq_MC : ∀ a1 a2 b1 b2, eq a1 a2→  eq b1 b2 → eq (plus a1 b1) (plus a2 b2);
        (*+ compatible avec l'egalite*)
  }.
```

En utilisant cette structure de monoïde, nous pouvons définir la structure de semi-anneau commutatif :

```
Record CSR (R : Type) (zero : R) (one : R) (plus : R → R → R) (mult : R → R → R) (TR : total_order R) : Type :=
mk_CS
  {
    one_zero_SR : not (eq zero one); (*0 different de 1*)
    isMC_plus_SR : CM zero plus TR; (*(A, 0, +, TR) est un monoide commutatif*)
    isMC_mult_SR : CM one mult TR; (*(A, 1, x, TR) est un monoide commutatif*)
    mult_distr_plusSR : ∀ a1 a2 b, eq (mult b (plus a1 a2)) (plus (mult b a1) (mult b a2)); (*x distributif sur +*)
    mult_zero_absorb_leftSR : ∀ a, eq (mult a zero) zero (*0 absorbant pour x a droite*)
  }.
```

La propriété de commutativité permet d'éliminer certaines règles redondantes comme $0 \times a = 0$.

Dans les sous-sections suivantes, nous instancions cette structure de semi-anneau commutatif avec des objets mathématiques fréquemment utilisés en provenance.

3.2 Exemples de semi-anneaux commutatifs

Nous donnons dans cette sous-partie quelques exemples de semi-anneaux de provenance, et leur formalisation.

3.2.1 Entiers naturels

Annoter les données avec les entiers naturels est un moyen concis de donner leur nombre d'occurrences dans une table. Nous verrons par la suite que cela permet de relier la notion de provenance à l'interprétation usuelle des bases de données.

Dans une démarche d'efficacité d'exécution, nous avons choisi la représentation de la librairie standard `N`. L'ordre choisi est l'ordre naturel sur les entiers naturels.

3.2.2 Booléens

Les booléens sont employés pour signifier la présence ou l'absence d'un n-uplet dans une base de données selon si ce n-uplet satisfait un prédicat. C'est l'exemple le plus simple possible pour les annotations. Il est peu utile en pratique, mais des généralisations peuvent être très intéressantes, comme par exemple la logique à trois valeurs pour représenter l'incertitude de la présence ou non d'un n-uplet.

Pour les booléens, quatre choix s'offrent à nous. D'une part, le choix de 1 ($\top$ ou $\bot$) impose le 0 ($\neg 1$) et la multiplication ($\wedge$ ou $\vee$). Dans tous les cas il reste deux choix possibles pour l'addition. Le tableau suivant résume les différentes possibilités.

1	**0**	$\times$	$+$
$\top$	$\bot$	$\wedge$	$\vee$
			$\oplus$
$\bot$	$\top$	$\vee$	$\wedge$
			$\iff$

Les booléens ne comportent pas d'ordre naturel entre les deux éléments $\bot$ et $\top$. Nous avons donc encore un degré de liberté supplémentaire pour enrichir cette structure avec un ordre total : soit $\top < \bot$ soit $\bot < \top$. Les deux ordres possibles étant symétriques, le choix de l'un par préférence à l'autre n'importe pas.

3.2.3 Entiers tropicaux

La troisième instance, beaucoup plus riche, est celle des entiers tropicaux. Les entiers tropicaux peuvent représenter par exemple le coût d'obtention d'une donnée, ou en inversant l'ordre naturel sur les entiers, des niveaux de confidentialité.

Nous notons $\mathbb{N}^\infty = \mathbb{N} \cup \{+\infty\}$ et rappelons la définition des entiers tropicaux :

Définition 7 (Semi-anneau des entiers tropicaux)**.** Le semi-anneau des entiers tropicaux est le semi-anneau $(\mathbb{N}^\infty, \infty, 0_\infty, min_\infty, +_\infty)$ tel que :

— $\forall a, b \in \mathbb{N}^2, min_\infty(a, b) = min(a, b)$
— $\forall a \in \mathbb{N}^\infty, min_\infty(a, \infty) = min_\infty(\infty, a) = a$
— $\forall a, b \in \mathbb{N}^2, a +_\infty b = a + b$
— $\forall a \in \mathbb{N}^\infty, a +_\infty \infty = \infty +_\infty a = \infty$

Afin d'être le plus général possible, plutôt que de construire uniquement le semi-anneau des entiers tropicaux, nous proposons de construire une formalisation générique d'un monoïde enrichi de l'infini, du minimum et d'une règle de compatibilité de l'addition, puis de montrer que cette extension correspond à un semi-anneau commutatif.

Définition 8 (Compatibilité avec l'addition à gauche)**.** $\forall a_1, a_2, b \in M^3$, si $a_1 < a_2$ alors on a soit : $b + a_1 = b + a_2$, soit : $b + a_1 < b + a_2$.

Nous formalisons les monoïdes-compat, c'est-à-dire les monoïdes ayant cette propriété, de la façon suivante :

```
Record CMcompat (M : Type) (zero : M) (plus : M → M → M) (TM : total_order M) : Type :=
mk_CMc
{
  isCM : CM zero plus TM; (*monoide*)
  equiv_plus_compat_CM_distr_lt : ∀ a1 a2 b, (*compatibilite*)
      lt TM a1 a2 → (eq TM (plus b a1) (plus b a2) ⋁ lt TM (plus b a1) (plus b a2));
}.
```

En utilisant la structure de monoïde muni d'un ordre total, nous pouvons définir la structure de semi-anneau étendu avec l'infini et le minimum :

```
Inductive MI := (* Ajout d'un element infini *)
| Infinity : MI
| EltM : M → MI.

Definition compMI a1 a2 := (*Extension de la comparaison avec l'infini : borne superieure*)
  match a1 with
  | Infinity ⇒
    match a2 with
    | Infinity ⇒ Eq
    | EltM b2 ⇒ Gt
    end
  | EltM b1 ⇒
    match a2 with
    | Infinity ⇒ Lt
    | EltM b2 ⇒ (t_compare TM) b1 b2
    end
  end.

Definition mulMI a1 a2 := (*Extension de l'addition avec l'infini (cela sera la multiplication du semi−anneau)*)
  match a1 with
  | Infinity ⇒ Infinity
  | EltM b1 ⇒
    match a2 with
    | Infinity ⇒ Infinity
    | EltM b2 ⇒ EltM (plus b1 b2)
    end
  end.

Definition minMI a1 a2 := (*Minimum (cela sera l'addition du semi−anneau)*)
  match compMI a1 a2 with
  |Gt ⇒ a2
  |_ ⇒ a1
  end.
```

Nous prouvons enfin le résultat suivant :

Théorème 1 (Monoïde-compat étendu avec l'infini)**.** Un monoïde-compat présentant un ordre total, peut être étendu avec l'infini et le minimum ; cette extension forme un semi-anneau commutatif.

Cette proposition nous permet d'ajouter de manière systématique, un infini et un minimum à une structure simple. Ainsi, pour prouver le caractère de semi-anneau commutatif des entiers tropicaux, il suffit donc de montrer que la structure des entiers naturels munie de l'ordre naturel est un monoïde-compat.

4 K-Algèbre en Coq

Nous souhaitons formaliser la sémantique de l'algèbre relationnelle annotée définie dans la section 2.

4.1 Contexte : formalisation de l'algèbre relationnelle

Nous nous inscrivons dans la continuité de l'article [4] qui formalise la syntaxe et la sémantique de l'algèbre relationnelle des bases de données sans annotations. Nous utilisons donc le type inductif `query` proposé pour la syntaxe des requêtes.

```
Inductive query : Type :=
  | Q_Empty_Tuple : query (*Relation contenant un n-uplet vide*)
  | Q_Table : relname → query (*Relation identifiee par un nom*)
  | Q_union : query → query → query (*Union*)
  | Q_Join : query → query → query (*Jointure naturelle*)
  | Q_Pi : list select → query → query (*Projection*)
  | Q_Sigma : q_formula → query → query (*Selection*)
```

Dans cette formalisation, `relname` est un type abstrait représentant l'ensemble des noms de relations. Une preuve de concept instancie ce type par le type `string` des chaînes de caractères, comme usuel en SQL.

La sémantique de l'algèbre relationnelle proposée dans [4] est définie par une fonction récursive `eval_query_rel` qui à chaque requête associe une relation. Les relations sont formalisées par des multi-ensembles de n-uplets. Pour cela, nous utilisons la bibliothèque des multiensembles finis `Febag`, qui se place dans le prolongement de Coccinelle [10].

```
Fixpoint eval_query_rel q : Febag.bag :=
  match q with
  | Q_Empty_Tuple ⇒ Febag.singleton empty_tuple (*multi-ensemble contenant un n-uplet vide*)
  | Q_Table r ⇒ instance_rel r
  | Q_union q1 q2 ⇒
    if sort q1 =?^{set} sort q2
    then Febag.union (eval_query_rel q1) (eval_query_rel q2) (*union de deux multi-ensembles*)
    else Febag.empty
  | Q_Join q1 q2 ⇒ natural_join_bag (eval_query_rel q1) (eval_query_rel q2)
                   (*Resultats de la jointure naturelle sur les multi-ensembles*)
  | Q_Pi s q ⇒ Febag.map (fun t ⇒ projection t s) (eval_query_rel q)
  | Q_Sigma f q ⇒ Febag.filter (fun t ⇒ eval_q_formula t f) (eval_query_rel q)
  end.
```

La fonction `instance_rel` associe au nom (unique) de la relation, le multi-ensemble des n-uplets contenus dans la relation.

La fonction `Febag.map` applique une fonction sur un multi-ensemble. Dans notre cas, elle applique la fonction `projection` qui renvoie la projection de t sur la sorte s. La fonction `sort` permet de donner les attributs pour une requête donnée. La fonction `Febag.filter` permet de ne conserver que les éléments d'un multi-ensemble qui satisfont un prédicat. Ici notre prédicat évalue une formule logique f sur un n-uplet t à travers la fonction `eval_q_formula`.

Dans le cas de l'union, si les sortes (les ensembles des attributs) sont bien les mêmes pour les deux sous-requêtes alors il est renvoyé l'union des deux multi-ensembles (en conservant les doublons éventuels). Si cela n'est pas le cas, il est renvoyé le multi-ensemble vide.

La formalisation de nos K-relations se base sur la syntaxe des requêtes précédentes.

4.2 Formalisation des K-relations

Dans cette sous-section, nous formalisons les K-relations et l'algèbre relationnelle pour la provenance.

Une K-relation est formalisée par le type `tuple → K` où `tuple` correspond à l'ensemble des n-uplets possibles et K est un semi-anneau commutatif. K est donc de type `CSR`, et nous noterons pour simplifier ses champs `zero, one, ...` et ainsi de suite. A partir de cela, nous définissons la sémantique de l'algèbre relationnelle sous la forme d'une fonction récursive `eval_query_prov` qui prend une requête et renvoie la K-relation correspondante.

```
Fixpoint eval_query_prov q : (tuple → K) :=
  match q with
  | Q_Empty_Tuple⇒ fun t ⇒ if eq_bool t empty_tuple then one else zero
  | Q_Table r ⇒ instance_prov r
  | Q_union q1 q2 ⇒
    if sort q1 =?set sort q2
    then fun t ⇒ plus ((eval_query_prov q1) t) ((eval_query_prov q2) t)
    else fun t ⇒ zero
  | Q_Join q1 q2 ⇒
    fun t ⇒
      mul
        (mul
          ((eval_query_prov q1) (mk_tuple T (sort q1) (dot t))) (*annotation de la projection du n−uplet sur les attributs
                de q1*)
          ((eval_query_prov q2) (mk_tuple T (sort q2) (dot t)))) (*annotation de la projection du n−uplet sur les attributs
                de q1*)
        (if support t =?set (sort q1 ∪ sort q2) then one else zero)
  | Q_Pi s q ⇒
  fun t' ⇒ finite_sum
                (List.map (eval_query_prov q)
                    (List.filter (fun t ⇒ eq_bool t' (projection t s) && negb (eq_bool ((eval_query_prov q) t) zero))
                        (content_of_query q)))
  | Q_Sigma f q ⇒ fun t ⇒ mul (
      match (eval_q_formula t f) with
      | true ⇒ one
      | false ⇒ zero
      end) ((eval_query_prov q) t)
    end
  end
```

Dans le cas du n-uplet vide, la fonction renvoyée vérifie si le n-uplet est équivalent au n-uplet vide et donne l'annotation un si c'est le cas et zéro sinon. Pour la table, la fonction `instance_prov` associe au nom de la relation, la K-relation correspondante. Dans le cas de l'union, il est vérifié que les sortes sont bien égales dans un premier temps. Si cela est le cas, il est retourné la fonction qui effectue l'addition des annotations des deux relations pour un n-uplet donné. Si cela n'est pas le cas, il est renvoyé la fonction constante égale à zéro. Pour la jointure naturelle, il faut que la fonction multiplie l'annotation de la projection du n-uplet sur les attributs de la première sous-requête avec l'annotation de la projection du n-uplet sur les attributs de la deuxième sous-requête mais aussi vérifie si le support du n-uplet est égal à l'union des sortes des deux requêtes. Dans le cas de la projection, il est associé à chaque n-uplet t la somme des annotations des n-uplets qui correspondent à la projection de t sur une sélection d'attributs s. La somme s'effectue à l'aide de la fonction `finite_sum` qui prend une liste d'éléments de K. Pour parcourir tous les n-uplets possibles dont la projection sur s est équivalente à t, nous avons besoin d'introduire l'ensemble des n-uplets contenus dans la sous-requête qui est formalisée par la fonction `content_of_query`. Cet ensemble ne contient qu'une seule occurrence de chaque n-uplet. Pour récupérer les bons n-uplets, la fonction `eval_query_prov` filtre les éléments de `content_of_query` dont la projection équivaut à t et vérifie que l'annotation de ces éléments n'est pas zéro. Il reste donc

à sommer les annotations de ces éléments. Enfin, la sélection renvoie une fonction qui dans un premier temps évalue la formule logique f sur un n-uplet avec la fonction `eval_q_formula`. Si le résultat est vrai alors nous renvoyons l'évaluation de la sous requête sur le n-uplet et sinon nous renvoyons zéro.

4.3 Propriétés générales

Dans cette sous-section, nous proposons quelques propriétés générales qui découlent de nos définitions.

Si nous supposons que l'annotation d'un certain n-uplet pour une requête n'est pas égale à zéro, nous pouvons prouver que ce n-uplet apparait bien dans la relation.

```
Lemma mem_eval_content : ∀ (q: query) (t: tuple),
    not (eq ((eval_query_prov q) t) zero) →
    mem_bool t (content_of_query q) = true.
```

Par ailleurs, nous prouvons également que si un n-uplet est dans une relation, le support de ce n-uplet est égal à la sorte de la requête.

```
Lemma in_content_support_sort: ∀ (q:query) (t:tuple),
      In t (content_of_query q) →
      support t [set]≡ sort q.
```

Avec les deux lemmes précédents, nous pouvons enfin prouver que si l'annotation pour un certain n-uplet n'est pas égale à zéro alors le support de cet n-uplet est égal à la sorte de la requête.

```
Lemma not_zeroK_eq_support_sort : ∀ (q:query) (t:tuple),
    not (eq ((eval_query_prov q) t) zero) →
    support t [set]≡ sort q.
```

5 Théorème d'adéquation

Nous pouvons remarquer qu'avoir des multi-ensembles de n-uplets et une fonction comptant le nombre d'occurrences des n-uplets sont deux façons analogues de définir des multi-ensembles.

Autrement dit, si nous prenons les relations et les K-relations correspondantes ayant pour annotation le nombre d'occurrences de chaque n-uplet, nous devons avoir une égalité entre le nombre d'occurrences d'un n-uplet lors de l'évaluation d'une requête q et l'annotation associée à ce n-uplet pour cette même requête q. Le semi-anneau approprié pour ces K-relations est celui des entiers naturels.

Pour avoir cette égalité, il est nécessaire de faire coïncider le cas de l'opérateur `Q_table`, c'est-à-dire que pour une relation nommée r, le nombre d'occurrences d'un n-uplet t dans le multi-ensemble `instance_rel r` doit être égal à l'annotation `instance_prov r t`. Nous posons donc l'hypothèse suivante :

```
Hypothesis instance_prov_nb_occ : ∀ (r : relname), (t : tuple),
          instance_prov r t = nb_occ t (instance_rel r).
```

La fonction `nb_occ` compte le nombre d'occurrences d'un n-uplet dans un multi-ensemble.

Pour prouver l'égalité pour toutes les requêtes, nous effectuons une induction sur la structure des requêtes. Le cas de la projection est le plus délicat à prouver. Cela nécessite de prouver que la somme finie de chaque annotation pour chaque n-uplet obtenue par la projection d'un n-uplet t est égale à une somme de chaque occurrence de chaque n-uplet obtenue par la projection d'un n-uplet t.

Nous introduisons une fonction auxiliaire pour prouver cette propriété.

```
Fixpoint map_reduce (e:tuple) (f : tuple → tuple) (g : tuple → N) (l : list (tuple)) :=
match l with
  | nil ⇒ 0
  | cons a l' ⇒ (if eq_bool e (f a) then 1 else 0) ∗ (g a)) + (map_reduce e f g l')
end.
```

Pour chaque n-uplet `e`, `map_reduce` parcourt la liste `l` et pour chaque élément `a` de `l`, elle vérifie s'il y a équivalence entre `e` et `f a` et si c'est le cas, ajoute `(g a)`.

Cette fonction nous permet de faire le pont entre l'évaluation dans l'algèbre relationnelle classique et dans la $\mathbb{N}$-algèbre. En effet, dans un premier temps, nous relions cette fonction à l'évaluation d'une part dans l'algèbre, d'autre part dans la $\mathbb{N}$-algèbre :

```
Lemma nb_occ_rel_map_reduce : ∀ (e:tuple),
    nb_occ e (Febag.map (fun t : tuple ⇒ projection t s) (eval_query_rel q)) =
    map_reduce e (fun t : tuple ⇒ projection t s) (fun _: tuple ⇒ 1) (Febag.elements (eval_query_rel q)).
```

et

```
Lemma finite_sum_map_reduce : ∀ (e:tuple),
  finite_sum
    (map (eval_query_prov_N q)
       (filter
          (fun t : tuple ⇒ eq_bool e (projection t s) && negb (eq_bool (eval_query_prov_N q t) 0))
                (content_of_query q))) =
  map_reduce e (fun t : tuple ⇒ projection t s)
    (fun t : tuple ⇒ nb_occ t (Febag.elements (eval_query_rel q)))
    (filter (fun x : tuple ⇒ negb (eq_bool (eval_query_prov_N q x) 0)) (content_of_query q)).
```

Dans un deuxième temps, nous voulons donc prouver que :

```
map_reduce e (fun t : tuple ⇒ projection t s) (fun _: tuple ⇒ 1) (Febag.elements (eval_query_rel q)) =
map_reduce e (fun t : tuple ⇒ projection t s)
    (fun t : tuple ⇒ nb_occ t (Febag.elements (eval_query_rel q)))
    (filter (fun x : tuple ⇒ negb (eq_bool (eval_query_prov_N q x) 0)) (content_of_query q))
```

Nous remarquons que la fonction `f` est la même dans les deux cas. Les deux listes correspondent aux n-uplets contenus dans la relation. Dans le cas des multi-ensembles, les n-uplets de la relation sont dans le multi-ensemble de `eval_query_rel q`. Pour les annotations, ces n-uplets correspondent aux n-uplets contenus dans `content_of_query` qui possèdent une annotation non nulle. Comme remarqué précédemment, `content_of_query` ne contient qu'une seule occurrence de chaque élément alors que le multi-ensemble peut en contenir plusieurs. La difficulté réside donc dans le fait que si nous avons un n-uplet `a` qui vérifie que `eq_bool e (f a)`, pour les multi-ensembles chaque occurrence est ajoutée, une à une, par la fonction constante égale à 1 alors que pour les annotations, le nombre d'occurrences de `a` est ajouté au plus une seule fois.

Pour cela, nous avons prouvé le lemme suivant :

```
Lemma equal_map_reduce_prov_rel : ∀ (e:tuple) (f:tuple→ tuple) (l1:list tuple) (l2 :list tuple),
    (∀ a1 a2, eq a1 a2 →
                  eq (f a1) (f a2)) →
    (∀ t, nb_occ t l1 =0 ⟷ nb_occ t l2 = 0) →
    (∀ t,
       nb_occ t l2 = 0 ⋁ nb_occ t l2 = 1) →
    map_reduce e f (fun t ⇒ nb_occ t l1) l2 =
    map_reduce e f (fun t ⇒ 1) l1.
```

Les hypothèses proposées supposent que la fonction projection est compatible avec l'équivalence, que si un n-uplet `t` n'est pas dans l'une des listes, il n'est pas dans l'autre ainsi que la liste `l2` a au plus une occurrence de chaque élément. La première hypothèse est raisonnable car la projection de deux n-uplets équivalents doit donner deux projections équivalentes. Les deux autres hypothèses découlent des remarques précédentes.

Avec ce lemme, nous avons obtenu l'égalité dans le cas de la projection. Les autres cas présentaient moins de subtilité. Nous avons donc prouvé notre théorème fondamental : [2]

```
Theorem K_relations_extend_relational_algebra :
  ∀(q:query),
    ∀(t:tuple), (eval_query_prov q) t = nb_occ t (eval_query_rel q).
```

ce qui se traduit par :

Théorème 2. Les K-relations sont une extension de l'algèbre relationnelle.

Autrement dit, en instanciant K par le semi-anneau commutatif des entiers naturels, nous retrouvons la sémantique de l'algèbre relationnelle.

6 Etat de l'art

Sur le versant de la formalisation de modèles, langages ou systèmes centrés données, des contributions ont vu le jour. Ainsi, la première tentative de formalisation, en Agda [24], de l'algèbre relationnelle est proposée par [13, 12] tandis que la première formalisation, complète, du modèle relationnel est proposée par [4] où le modèle de données, l'algèbre, les requêtes «tableaux», la procédure de semi-décision «chase» et les contraintes d'intégrité sont formalisés.

La toute première tentative de vérifier, en Coq, un SGBDR est présentée dans [20]. Des propositions de sémantiques mécanisées pour SQL sont présentées dans [2, 3]. Une formalisation d'un moteur de requêtes SQL est fournie dans [5].

À notre connaissance, il n'existe pas de travaux sur la formalisation au moyen d'assistants à la preuve de la provenance des données. Une formalisation des K-relations est donnée dans l'article [8] proposant un outil pour décider de l'équivalence de deux requêtes SQL. À cette fin, HottSQL, une sémantique pour un fragment de SQL est définie. Cette sémantique se base sur la notion de K-relation. Toutefois, la modélisation des K-relations proposée *relaxe la contrainte de finitude* de celles ci. Ainsi, cette propositon peut conduire à manipuler des instances de K-relations potentiellement infinies. En outre, la sémantique présentée n'est pas *exécutable*. Notre approche permet d'avoir une sémantique exécutable, ce qui offre la possibilité d'extraire du code, d'exécuter des tests, etc.

Sur le versant de la formalisation de structures algébriques, de nombreuses propositions ont été développées en Coq, la plus élaborée étant Mathematical Components [23]. Notre formalisation des semi-anneaux repose sur la bibliothèque Coccinelle [10] pour deux raisons : la compatibilité avec la formalisation de l'algèbre relationnelle sur laquelle repose ce travail, et, de nouveau, la volonté de fournir une sémantique exécutable. Sur ce deuxième aspect, nous envisageons de nous lier à Mathematical Components via le mécanisme de raffinements proposé par Cohen *et al.* [9].

7 Conclusions et perspectives

Nous avons présenté les bases pour la formalisation de la provenance des données. En nous basant sur des formalisations existantes de l'algèbre relationnelle d'une part, et des structures algébriques usuelles d'autre part, nous avons développé le modèle de provenance basé sur les K-relations. Notre avons enfin prouvé le théorème fondamental montrant l'adéquation de cette

2. Dans la formalisation disponible en ligne, les requêtes contiennent des agrégats qui ne sont pas encore traités dans notre formalisation de la provenance.

généralisation avec l'algèbre relationnelle usuelle. Notre formalisation offre un niveau d'abstraction très élevé, et est notamment paramétrique en le semi-anneau de provenance. Elle est développée dans l'assistant de preuve Coq.

Notre prochaine étape est d'étendre les K-algèbres au cas des agrégats comme proposé dans [1], et ainsi avoir une sémantique pour les annotations pour les agrégats mécanisée et exécutable en Coq. La combinaison dans le cas des agrégats repose sur la notion de semi-module, qui permet d'agréger des annotations (comme les agrégats agrègent les données).

Nous souhaitons lier ce travail à la formalisation du langage de requêtes SQL et à sa traduction certifiée vers l'algèbre relationnelle [3]. Pour cela, nous voulons proposer un langage de requêtes tenant compte des annotations et les traduisant vers l'algèbre de manière correcte et complète.

À long terme, notre objectif est de certifier la provenance dans des analyses de données telles qu'utilisées en réseau, bio-informatique, etc. Notre cas d'étude sera les workflows scientifiques qui, à partir de données d'entrées et d'une suite d'instructions (pouvant contenir des boucles), génèrent de nouvelles données ensuite analysées. Ainsi, les workflows scientifiques seront instrumentés pour fournir une trace, dont l'outil certifié en Coq pourra garantir la reproductibilité : ce processus de certification *a posteriori* reposera sur une approche sceptique, et sera donc indépendante des outils apparaissant dans les workflows [18]. Pour mener à bien la vérification de ces traces, il faudra définir un langage formel de spécification de workflows, permettant d'établir des propriétés sur ces derniers et de montrer l'équivalence de (parties de) workflows, comme requis dans [6] pour garantir la reproductibilité. L'un des enjeux de cette phase sera d'assurer un passage à l'échelle au niveau du volume des données et des requêtes posées. Nous travaillons actuellement sur la constitution de cas d'utilisations réels pour évaluer plus précisément les capacités de notre solution sur ces aspects.

Remerciements : Nous remercions les rapporteurs anonymes pour leurs remarques pertinentes.

Références

[1] Yael Amsterdamer, Daniel Deutch, and Val Tannen. Provenance for aggregate queries. In *Proceedings of the thirtieth ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*, pages 153–164. ACM, 2011.

[2] J. S. Auerbach, M. Hirzel, L. Mandel, A. Shinnar, and J. Siméon. Handling environments in a nested relational algebra with combinators and an implementation in a verified query compiler. In *Proceedings of the 2017 ACM International Conference on Management of Data, SIGMOD Conference 2017, Chicago, IL, USA, May 14-19, 2017*, pages 1555–1569, 2017.

[3] Véronique Benzaken and Évelyne Contejean. A coq mechanised formal semantics for realistic SQL queries : formally reconciling SQL and bag relational algebra. In Assia Mahboubi and Magnus O. Myreen, editors, *Proceedings of the 8th ACM SIGPLAN International Conference on Certified Programs and Proofs, CPP 2019, Cascais, Portugal, January 14-15, 2019*, pages 249–261. ACM, 2019.

[4] Véronique Benzaken, Évelyne Contejean, and Stefania Dumbrava. A Coq Formalization of the Relational Data Model. In Zhong Shao, editor, *ESOP - 23rd European Symposium on Programming*, Lecture Notes in Computer Science, Grenoble, France, April 2014. Springer.

[5] Véronique Benzaken, Evelyne Contejean, Ch. Keller, and E. Martins. A coq formalisation of sql's execution engines. In Jeremy Avigad and Assia Mahboubi, editors, *Interactive Theorem Proving - 9th International Conference, ITP 2018, Held as Part of the Federated Logic Conference, FloC*

2018, Oxford, UK, July 9-12, 2018, Proceedings, volume 10895 of *Lecture Notes in Computer Science*, pages 88–107. Springer, 2018.

[6] Sarah Cohen Boulakia, Khalid Belhajjame, Olivier Collin, Jérôme Chopard, Christine Froidevaux, Alban Gaignard, Konrad Hinsen, Pierre Larmande, Yvan Le Bras, Frédéric Lemoine, Fabien Mareuil, Hervé Ménager, Christophe Pradal, and Christophe Blanchet. Scientific workflows for computational reproducibility in the life sciences : Status, challenges and opportunities. *Future Generation Comp. Syst.*, 75 :284–298, 2017.

[7] Peter Buneman, Sanjeev Khanna, and Wang Chiew Tan. Why and where : A characterization of data provenance. In Jan Van den Bussche and Victor Vianu, editors, *Database Theory - ICDT 2001, 8th International Conference, London, UK, January 4-6, 2001, Proceedings.*, volume 1973 of *Lecture Notes in Computer Science*, pages 316–330. Springer, 2001.

[8] S. Chu, K. Weitz, A. Cheung, and D. Suciu. HoTTSQL : Proving query rewrites with univalent SQL semantics. In *PLDI*. ACM, 2017.

[9] Cyril Cohen, Maxime Dénès, and Anders Mörtberg. Refinements for free ! In Georges Gonthier and Michael Norrish, editors, *Certified Programs and Proofs - Third International Conference, CPP 2013, Melbourne, VIC, Australia, December 11-13, 2013, Proceedings*, volume 8307 of *Lecture Notes in Computer Science*, pages 147–162. Springer, 2013.

[10] Évelyne Contejean. Coccinelle, a Coq library for rewriting. In *Types*, Torino, Italy, March 2008.

[11] Yingwei Cui, Jennifer Widom, and Janet L. Wiener. Tracing the lineage of view data in a warehousing environment. *ACM Trans. Database Syst.*, 25(2) :179–227, June 2000.

[12] C. Gonzalia. Towards a formalisation of relational database theory in constructive type theory. In *RelMiCS*, pages 137–148, 2003.

[13] C. Gonzalia. *Relations in Dependent Type Theory*. PhD thesis, Chalmers Göteborg University, 2006.

[14] Todd J. Green. Containment of conjunctive queries on annotated relations. In *Proceedings of the 12th International Conference on Database Theory*, ICDT '09, pages 296–309. ACM, 2009.

[15] Todd J. Green, Gregory Karvounarakis, and Val Tannen. Provenance semirings. In Leonid Libkin, editor, *Proceedings of the Twenty-Sixth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems, June 11-13, 2007, Beijing, China*, pages 31–40. ACM, 2007.

[16] Todd J Green, Grigoris Karvounarakis, and Val Tannen. Provenance semirings. In *Proceedings of the twenty-sixth ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*, pages 31–40. ACM, 2007.

[17] Todd J. Green and Val Tannen. The semiring framework for database provenance. In Emanuel Sallinger, Jan Van den Bussche, and Floris Geerts, editors, *Proceedings of the 36th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems, PODS 2017, Chicago, IL, USA, May 14-19, 2017*, pages 93–99. ACM, 2017.

[18] J. Harrison and L. Théry. A Sceptic's Approach to Combining HOL and Maple. *Journal of Automated Reasoning*, 21(3) :279–294, 1998.

[19] Donna Maglott, Jim Ostell, Kim D Pruitt, and Tatiana Tatusova. Entrez gene : gene-centered information at ncbi. *Nucleic acids research*, 33(suppl_1) :D54–D58, 2005.

[20] Gregory Malecha, Greg Morrisett, Avraham Shinnar, and Ryan Wisnesky. Toward a verified relational database management system. In *ACM Int. Conf. POPL*, 2010.

[21] Laurel Orr, Dan Suciu, and Magdalena Balazinska. Probabilistic database summarization for interactive data exploration. *PVLDB*, 10(10) :1154–1165, 2017.

[22] Kim D Pruitt, Tatiana Tatusova, Garth R Brown, and Donna R Maglott. Ncbi reference sequences (refseq) : current status, new features and genome annotation policy. *Nucleic acids research*, 40(D1) :D130–D135, 2011.

[23] The Mathematical Components team. Mathematical Components. Available at https://math-comp.github.io/math-comp.

[24] The Agda Development Team. *The Agda Proof Assistant Reference Manual*, 2010.