Opfibrations

CS 6118 - Types and Semantics

A SQL query such as `SELECT first + " " + last FROM Employees` can be viewed as a comprehension `for (e in Employees) e.first + " " + e.last`. We can formalize these constructs in a flexible manner by using what are known as opfibrations. This categorty theoretic construct is a natural fit for the construct, and taking such an abstract perspective guides us as to how we should interepret comprehension syntax in other variations of databases, such as allowing redundant entries, probabilistic entries, or incorporating other document information such as distances between words. First, to understand this perspective, we have to see how we can design morphisms so that they can represent query answers.

1 Tableau Theory

Suppose we have the following database instance:

`theatre`	`movie`	`time`
AMC	Spider-Man	6pm
AMC	Spider-Man	9pm
AMC	Metropolis	6pm
Cinemapolis	Babe	6pm
Cinemapolis	Metropolis	9pm
Cinemapolis	Babe	6pm

Now I'd like to see a movie at 6, but I might be late, so I only want to see a movie if it's also showing at that same theatre later at 9. You can think of this query as a database (or tableau):

`theatre`	`movie`	`time`
t	m	6pm
t	m	9pm

This says I want a theatre t and movie m such that m is playing at t at both 6pm and 9pm.

Now, call the above table Q, and suppose we have some database instance I that we want results from. A result is any assignment θ of t and m such that, for all entries e in Q, $e[\theta]$ (i.e. the entry e after replacing t and m with their assignments in θ) is in I. We call such a θ a (constant preserving) database homomorphism from Q to I. As such, we can formalize a category for databases, though for simplicity we will ignore constant preservation.

Definition (**Dat** - The Category of Databases)**.** Suppose we have some category **Sch** of schemas and schema morphisms, and we have some functor $[\![-]\!] : \mathbf{Sch} \to \mathbf{Set}$ specifying the semantics of each schema and schema morphism. Then define **Dat** as the category with

Objects $\langle \Sigma, D \rangle$**.** A schema Σ and a finite subset D of $[\![\Sigma]\!]$

Morphisms $\langle f, p \rangle : \langle \Sigma, D \rangle \to \langle \Sigma', D' \rangle$**.** A schema morphism $f : \Sigma \to \Sigma'$ and a proof p of $\forall d \in D.[\![f]\!](d) \in D'$

2 Opfibrations

There is an important functor from **Dat** to **Set** called the underlying functor:

Definition ($U : \mathbf{Dat} \to \mathbf{Set}$ - The underlying functor of **Dat**)**.**

$U(\langle \Sigma, D \rangle) = \Sigma$

$U(\langle f, p \rangle) = f$

This picks out the underlying structure of databases and database morphisms. Thus databases can be seen as structure laid over schemas. As we will see, in this particular case the overlaid structure forms an opfibration, which consequently gives a semantics to comprehensions.

Now let us reconsider the comprehension `for (e in Employees) e.first + " " + e.last`. `Employees` is a database, i.e. something from the overlaid world. λ`e. e.first + " " + e.last` on the other hand is a simple function, i.e. something from the underlying world. In particular, it is a function from the underlying structure of `Employees` to a new schema, namely just `string`. We expect the result R of this comprehension to be a database with schema `string` such that our function, call it `name`, applied to each entry in `Employees` produces an entry in R. More formally, we want a database R on scheme `string` and database homomorphism $f : \langle \texttt{Employees}, \texttt{employee} \rangle \to \langle R, \texttt{string} \rangle$ such that $U(f) = \texttt{name}$. This is called a lifting of `name`; we are taking a non-database function and lifting it into a database homomorphism.

Now, there can be many such liftings of `name`. Our expectation is that R has precisely the entries corresponding to applying `name` to the entries in `Employees`, no more, no less. The "no less" requirement corresponds to saying that `name` must be a database homomorphism from `Employees` to R. The "no more" requirement, though, requires more attention. We can think of this as saying that R must be the smallest database such that `name` is a database homomorphism from `Employees` to it. In other words, given any other database D such that `name` is a database homomorphism from `Employees` to D, R must be a subset of D. We can rephrase the latter clause as saying that the identity function from `string` to `string` must be a database homomorphism from R to D. This leads us to the concept of an opcartesian morphism.

Definition. Opcartesian Morphism Suppose we have some category (call it **Dat**) representing the overlaid structure, another category (call it **Sch**) representing the underlying structure, and a functor $U : \mathbf{Dat} \to \mathbf{Sch}$ mapping overlaid structure to its underlying structure. Suppose $\bar{f} : D \to E$ is a morphism in **Dat**. $\bar{f}$ is opcartesian if, for any morphism $g : U(E) \to U(F)$ in **Sch** and $\bar{h} : D \to F$ with $U(\bar{h}) = U(\bar{f})\,;g$, there exists a unique $\bar{g} : E \to F$ with $U(\bar{g}) = g$ and $\bar{f}\,;\bar{g} = \bar{h}$.

The intent of this definition is to convey that E has the minimal structure necessary to make $\bar{f}$ a lifting of $U(\bar{f})$. The idea is that, given any other g extending $U(\bar{f})$ such that the composition can be lifted (i.e. there is an $\bar{h}$ with underlying structure $U(\bar{f})\,;g$), then since E has so little structure it is guaranteed that g can be lifted as well (i.e. $\bar{g}$) to preserve that overlaid structure. Notice that, if g is the identity function, then $U(\bar{h})$ must be $U(\bar{f})$, i.e. another lifting of $U(\bar{f})$, so $\bar{f}$ being opcartesian means that there is a lifting of the identify function from E to F showing that E "fits inside" F.

Definition. Opfibration Suppose we have some category (call it **Dat**) representing the overlaid structure, another category (call it **Sch**) representing the underlying structure, and a functor $U : \mathbf{Dat} \to \mathbf{Sch}$ mapping overlaid structure to its underlying structure. This triple is an opfibration if, for every $f : U(D) \to X$ there is an E and opcartesian morphism $\bar{f} : D \to E$ with $U(\bar{f}) = f$.

The opcartesian lifting $\bar{f}$ of f is unique up to isomorphism, so the convention is to denote its codomain (E in the above definition) as $f_!(D)$.

Example: Dat Given a database $\langle \Sigma, D \rangle$ and a schema morphism f from $U(\langle \Sigma, D \rangle) = \Sigma$ to some Σ', define $f_!(D)$ as $\langle \Sigma', D' \rangle$ where D' is the subset $\{f(d) \mid d \in D\}$ or equivalently $\{d' \in D' \mid \exists d \in D.d' = f(d)\}$ both guaranteed to be finite since D is finite. Clearly f maps elements in D to elements in D', so we have a database homomorphism from $\langle \Sigma, D \rangle$ to $\langle \Sigma', D' \rangle$ which is a lifting of f.

To prove that this is an opcartesian lifting, suppose we have a schema morphism g from $U(f_!(D)) = \Sigma'$ to Σ'', and we have a database homomorphism from $\langle \Sigma, D \rangle$ to some $\langle \Sigma'', D'' \rangle$ lifting $f\,;g$, then we need to prove that g can be lifted to a database homomorphism from $\langle \Sigma', D' \rangle$ to $\langle \Sigma'', D'' \rangle$. So suppose d' is in D', then we need to show $g(d')$ is in D''. By definition of D', there must be a d in D such that $d' = f(d)$, so that we can alternatively show that $g(f(d))$ is in D''. Fortunately, since $f\,;g$ can be lifted to a database homomorphism and d is in D, by definition this means that $g(f(d))$ is in D'', thus proving that g can be lifted to a database homomorphism of the appropriate type.

Remark Notice that the definition of $f_!(D)$ is of the form "have only whatever structure D has" as made explicit by the use of the existential quantifier. $f_!$ essentially pushes out the structure of D onto the codomain of f, which is why it corresponds to comprehensions.

Example: Structured Documents Often not only the contents of a document but the relative positions of those contents are important for searching. For example, "trunk" has many meanings, but it is much more likely to have a specific meaning if it occurs close to the word "elephant". Thus if we wanted to adapt the tableau-model approach to queries to structured documents, we would make a structured document containing just the words "trunk" and "elephant" that are noted to be, say, 5 words apart. We would then want such a structured document to have a homomorphism to any other document where "trunk" and "elephant" occur 5 words *or fewer* apart. In particular, this means we want to allow contractive mappings.

Definition (**Met*** : The category of simplified metric spaces with contractive mappings)**.**

Objects $\langle X, d\rangle$**.** A set X and a function $d : X \times X \to [0, \infty]$

Morphisms $\langle f, p\rangle : \langle X, d\rangle \to \langle Y, d'\rangle$**.** A function $f : X \to X'$ and proof p of $\forall x, x' \in X. d'(f(x), f(x')) \leq d(x, x')$

Now suppose we have a simplified metric space $\langle X, d\rangle$ and a function $f : X \to Y$. Define $f_!(d)$ as $\langle Y, d'\rangle$ where $d'(y, y') = \min_{x,x' \in X | f(x)=y \wedge f(x')=y'} d(x, x')$. Again, clearly f can be lifted to a contractive mapping from $\langle X, d\rangle$ to $\langle Y, d'\rangle$. Notice this definition implicitly relies on the fact that ∞ is a valid distance, otherwise we could only define $f_!$ when f is surjective.

Now suppose we have a simplified metric space $\langle Z, d''\rangle$ and a function $g : Y \to Z$ such that $f\,;g$ can be lifted to a contractive mapping from $\langle X, d\rangle$ to $\langle Z, d''\rangle$. Then given y and y' in Y, $d''(g(y), g(y'))$ is less than or equal to $d(x, x')$ for all x and x' in X such that $y = f(x)$ and $y' = f(x')$ since $f\,;g$ is contractive, which implies $d''(g(y), g(y'))$ is less than or equal to $\min_{x,x' \in X | f(x)=y \wedge f(x')=y'} d(x, x')$ which is the definition of d'. Thus g can be lifted to a contractive mapping of the appropriate type, consequently implying that $f_!(d)$ gives an opcartesian lifting. This shows how comprehension syntax can be applied to simplified metric spaces.

Non-Example: Irreflexive Relations Irreflexive relations do not form an opfibration. They do occasionally have opcartesian liftings, but not always.

Definition (**Irrefl** : The category of irreflexive relations)**.**

Objects $\langle X, R, i\rangle$**.** A set X, a subset $R \subset X \times X$, and a proof i of $\forall x \in X. \neg(x\ R\ x)$

Morphisms $\langle f, p\rangle : \langle X, R, i\rangle \to \langle Y, S, j\rangle$**.** A function $f : X \to Y$ and proof p of $\forall x, x' \in X. x\ R\ x' \Rightarrow f(x)\ S\ f(x')$

Now consider the relation $\langle \{a, b\}, \{\langle a, b\rangle\}\rangle$ and the unique function f from $\{a, b\}$ to $\{c\}$. Not only is there no opcartesian lifting of f, there is no lifting of f at all. For f to be made relation-preserving, c would have to be related to itself, which would make the relation not irreflexive. The main problem is that f maps to distinct elements to the same element. For injective f, there is an opcartesian lifting. Thus, one could apply comprehension syntax to irreflexive relations *provided* the language is restricted so that the body of a comprehension is always injective.