An Introduction to Categories with Haskell and Databases

Ryan Holbrook

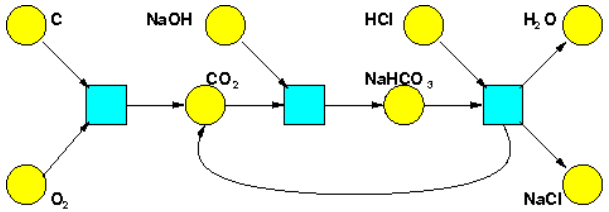
May 2, 2019

Outline

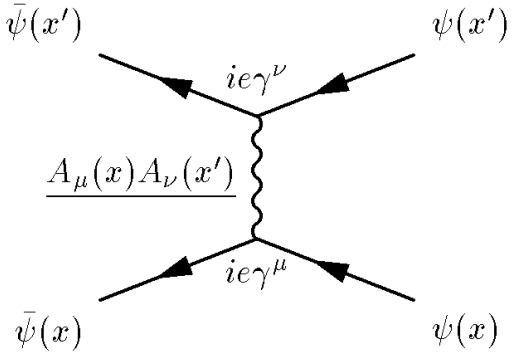
What is Category Theory?

- Category theory is the language of coherent, composable systems.
- It tells us how to build systems and how to translate from one system to another. (And even how to translate translations!)

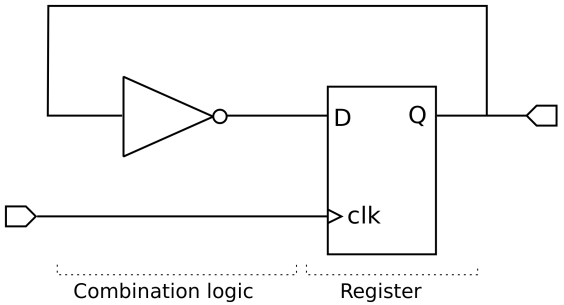
Chemistry



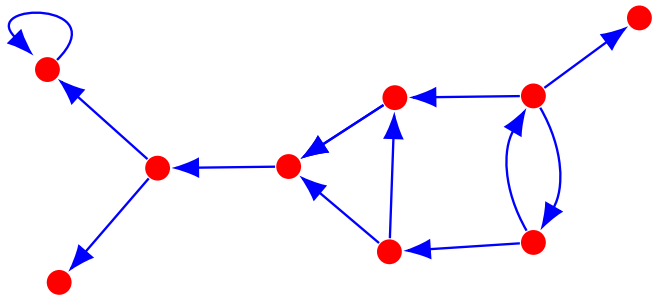
Quantum Physics

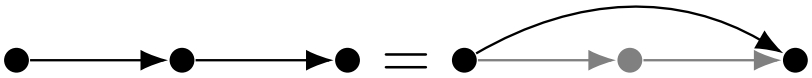


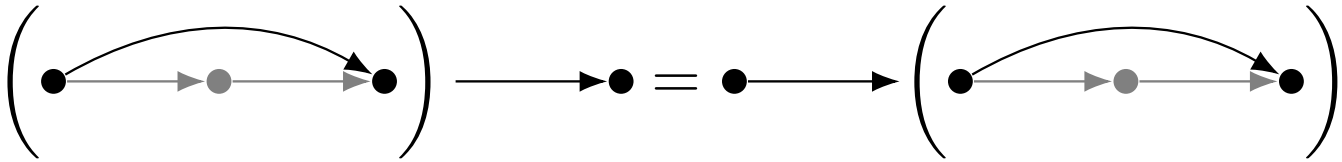
Electronics

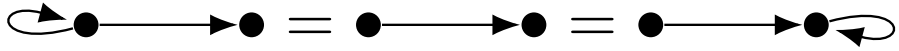


A *category* consists of objects and arrows.



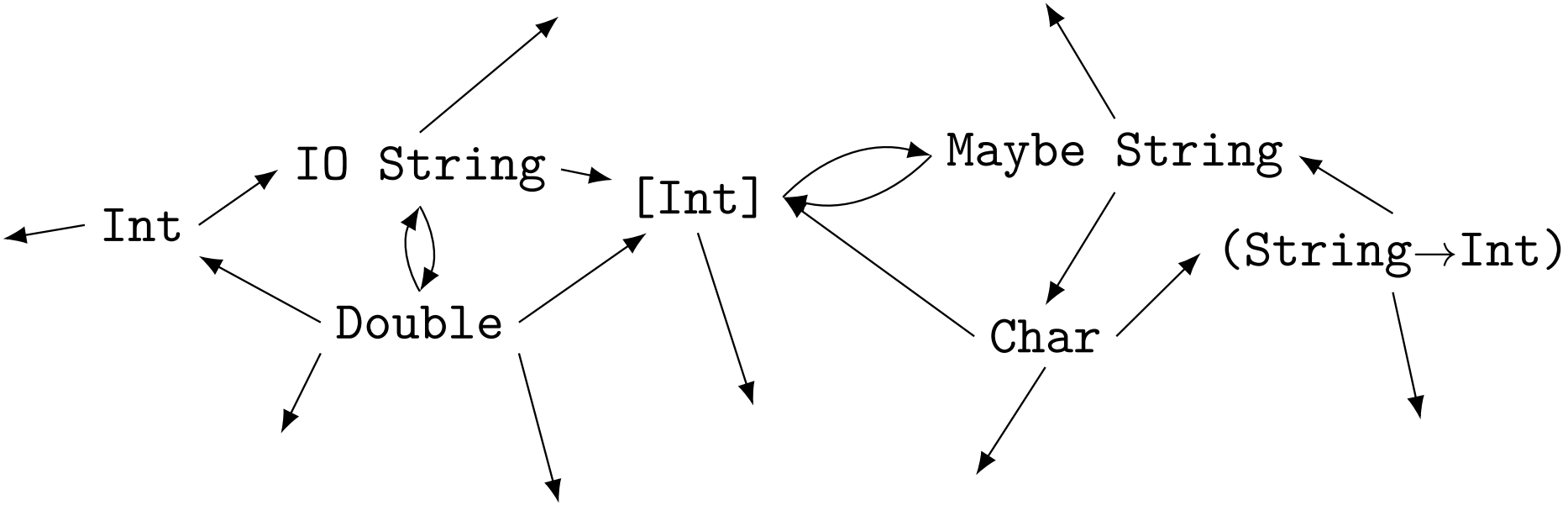
- Arrows compose: 
- And they do it coherently.
  - Associativity:



  - Identity: 

The Hask Category

The Haskell type system forms a category.[1]



Types are objects. Functions are arrows.

[1] Almost.

The Hask Category

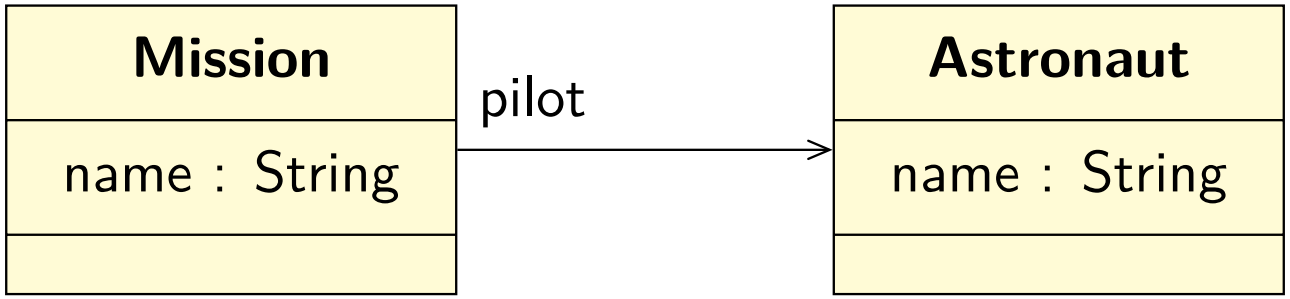
Types are objects. Functions are arrows.

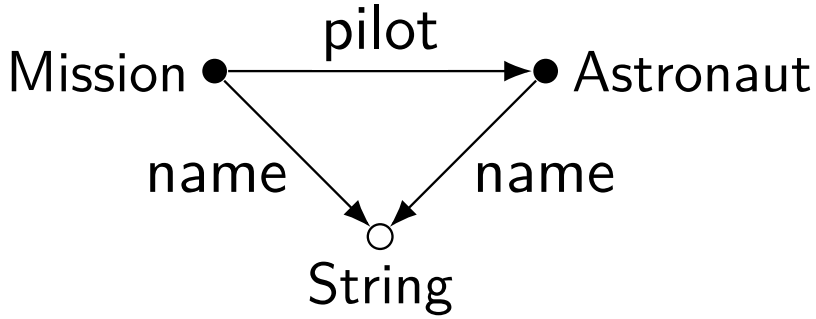
- Functions compose: `(f.g) x == f (g x)`
- Coherently:
  - Associativity: `(f.g).h $ x == f.(g.h) $ x`
  - Identity: `Id x == x`

So now we get to use functors, applicatives, monads... (!)

Databases are Categories

A database schema is a category.



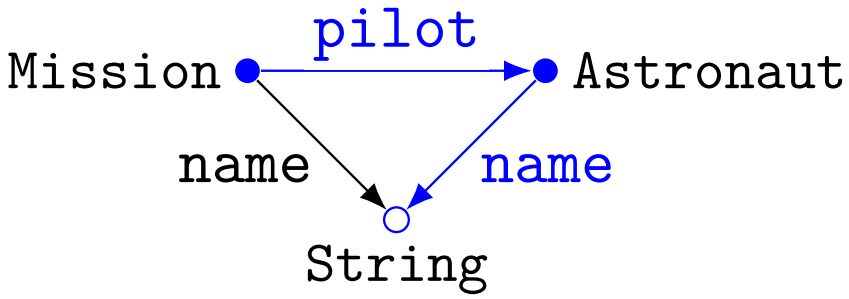


Tables are objects. Relations are arrows.

Databases are Categories

Composition is JOIN.

- Query: Find the name of every pilot that has flown on a mission.



SQL

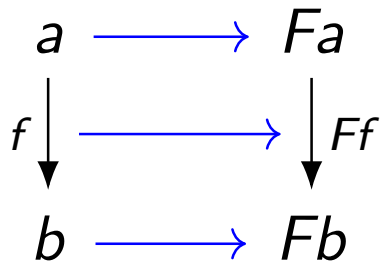
```
SELECT Astronaut.name
FROM Astronaut
INNER JOIN Mission ON
Astronaut.id = Mission.pilot
```

CQL

```
from
  m : Mission
attributes
  name -> m.pilot.name
```

What is a Functor?

A functor is a coherent mapping of categories.

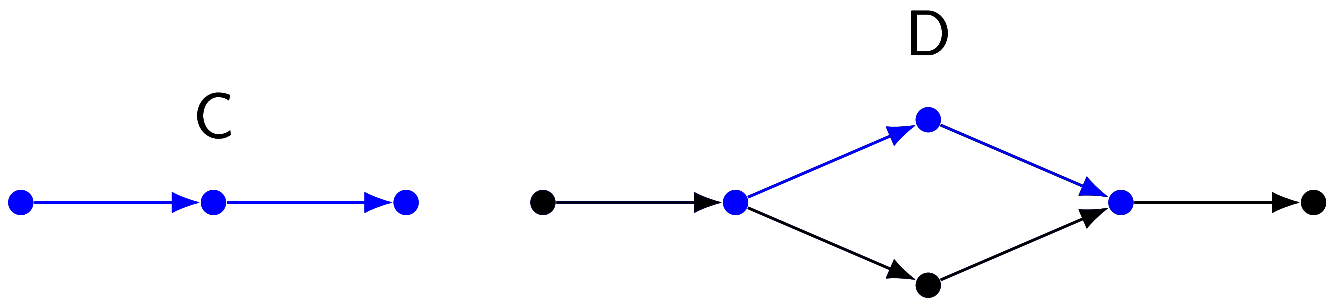


The mapping must preserve coherency:

- $F(g \circ f) = Fg \circ Ff$
- $F(Id) = Id$

What is a Functor?

A functor $F : C \to D$ makes an image of C inside of D.



Functors in Haskell

A functor is a type that implements `fmap`:

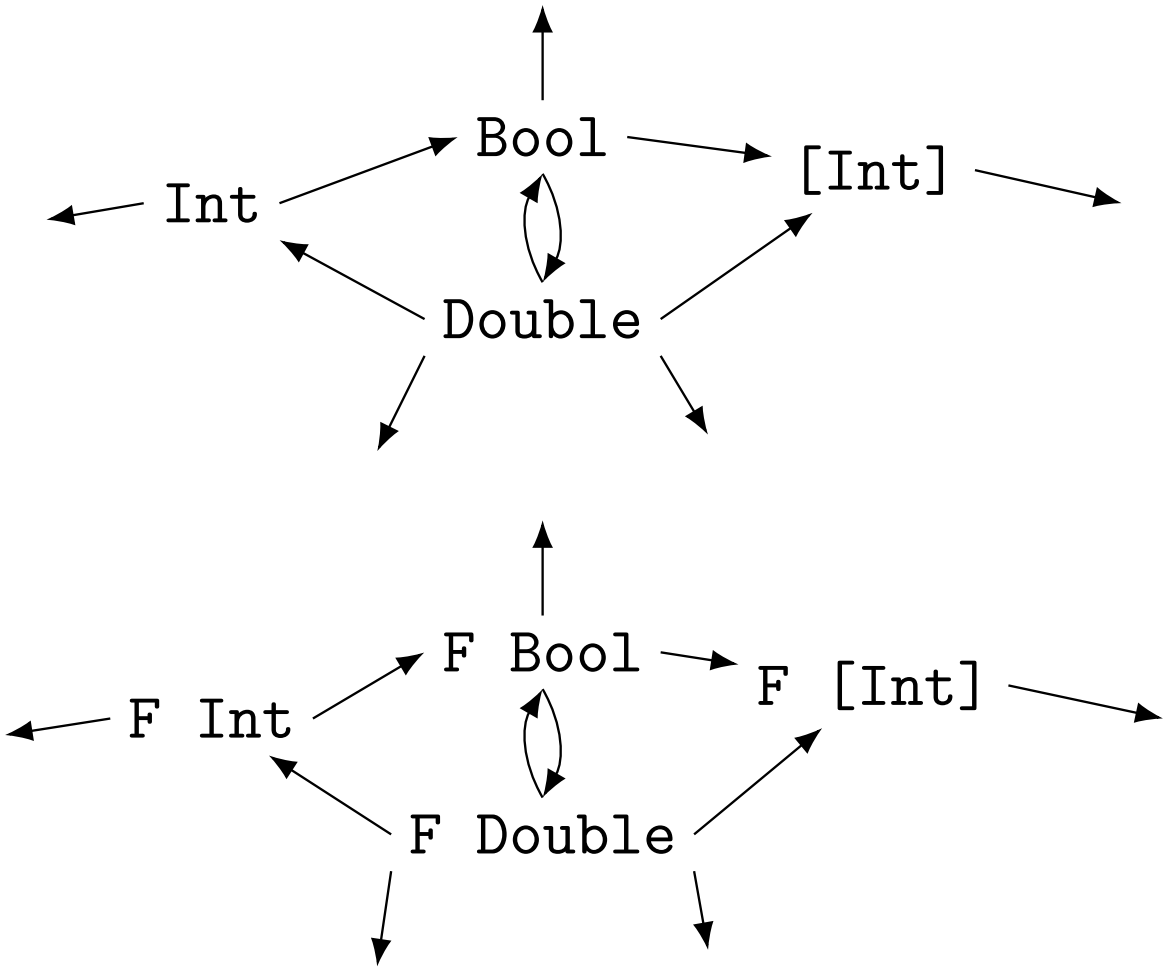
```
(<$>) :: (a -> b) -> F a -> F b
```

Here, `F` is the name of the functor. It must satisfy the two functor laws:

- `fmap id == id`
- `fmap (g.f) == fmap g . fmap f`

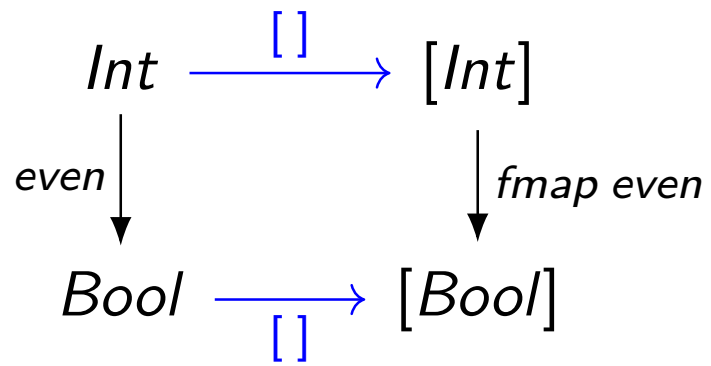
Functors in Haskell

A functor `F` makes an image of Hask inside of Hask.



Which must also contain its own image... It's functors all the way down!

Functors in Haskell: The List Functor



Example

```
> even 2
True
> fmap even [-2,-1,0,1,2]
[True,False,True,False,True]
```

Hask

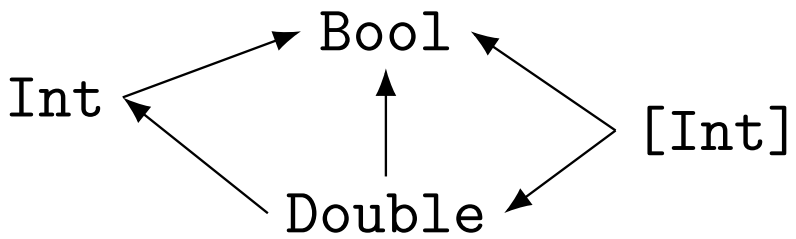
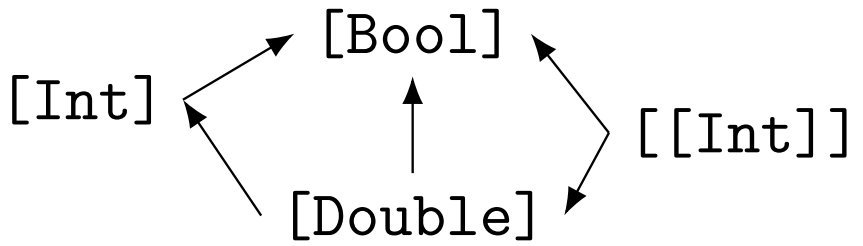


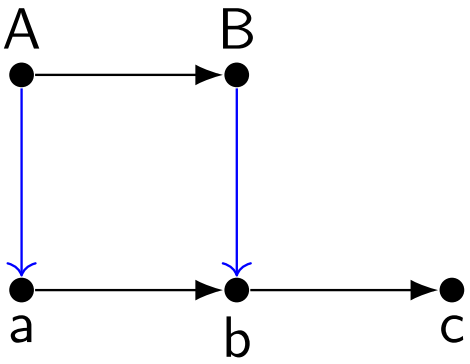
Image of List in Hask



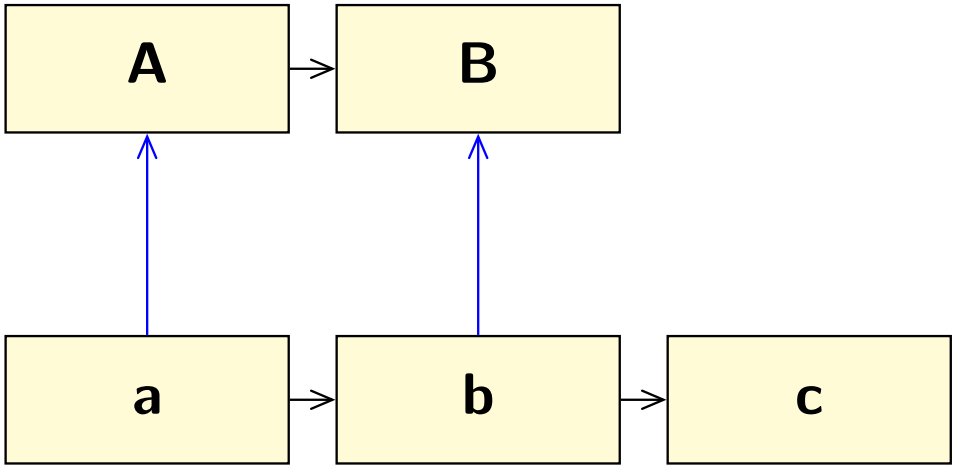
Database Queries as Functors

A query is a coherent mapping of schemas. The data returned has to obey the schema relations.

1. Define the mapping

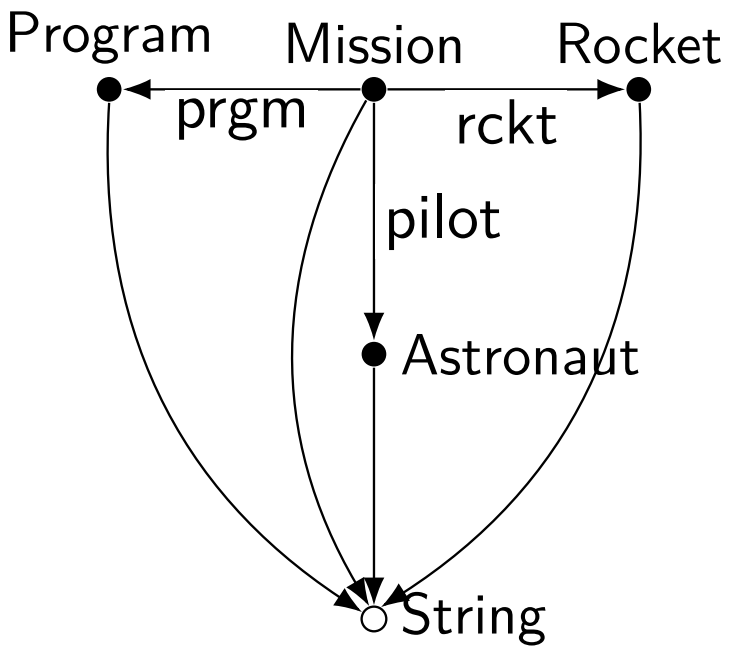


2. Pull data back with `eval`



Defining the Database

Space Agency Schema



Nasa Instance

Astronaut (4)

Row	name
0	**Edward H White II**
1	**Buzz Aldrin**
2	**John Glenn**
3	**Fred Haise**

Mission (5)

Row	name	pilot	program	rocket
4	**Apollo 11**	1	10	12
5	**Gemini 4**	0	9	13
6	**Mercury 6**	2	11	14
7	**Gemini 12**	1	9	13
8	**Apollo 13**	3	10	12

Program (3)

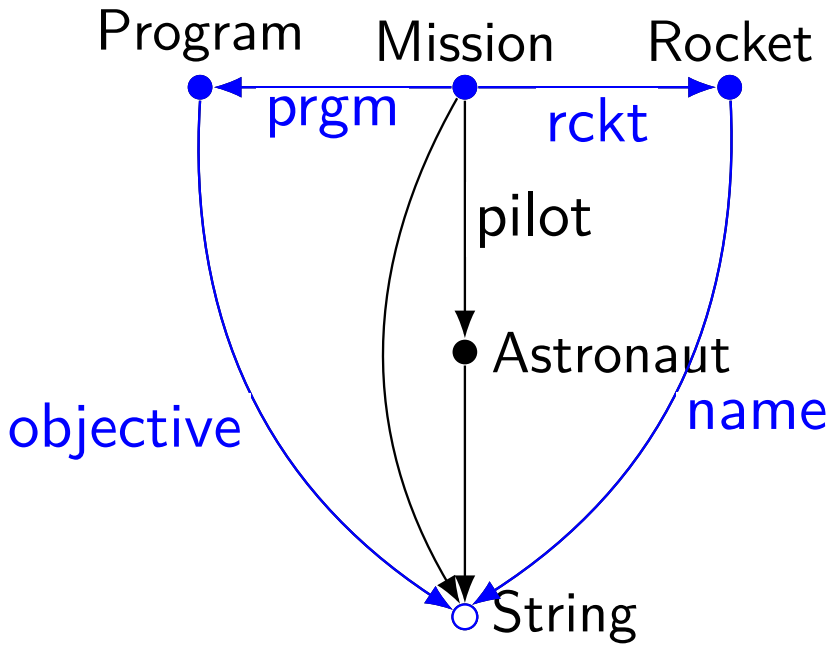
Row	name	objective
9	**Gemini**	**Extravehicular activity**
10	**Apollo**	**Manned lunar landing**
11	**Mercury**	**Manned Earth orbital flight**

Rocket (3)

Row	name
12	**Saturn V**
13	**Gemini-Titan II**
14	**Mercury-Atlas**

Defining a Query

Space Agency Schema



Query Schema

Query

The names of all rockets that have flown on a mission with a pilot named "Buzz" and the objective of the mission's program.

Example: A Space Agency

Query

```
query qBuzz = simple : SpaceAgency {
        from
          m : Mission
        where
          Matches(m.pilot.name, "Buzz.*") = true
        attributes
          rcktName -> m.rocket.name
          prgmObjective -> m.program.objective
}
```

Result Instance

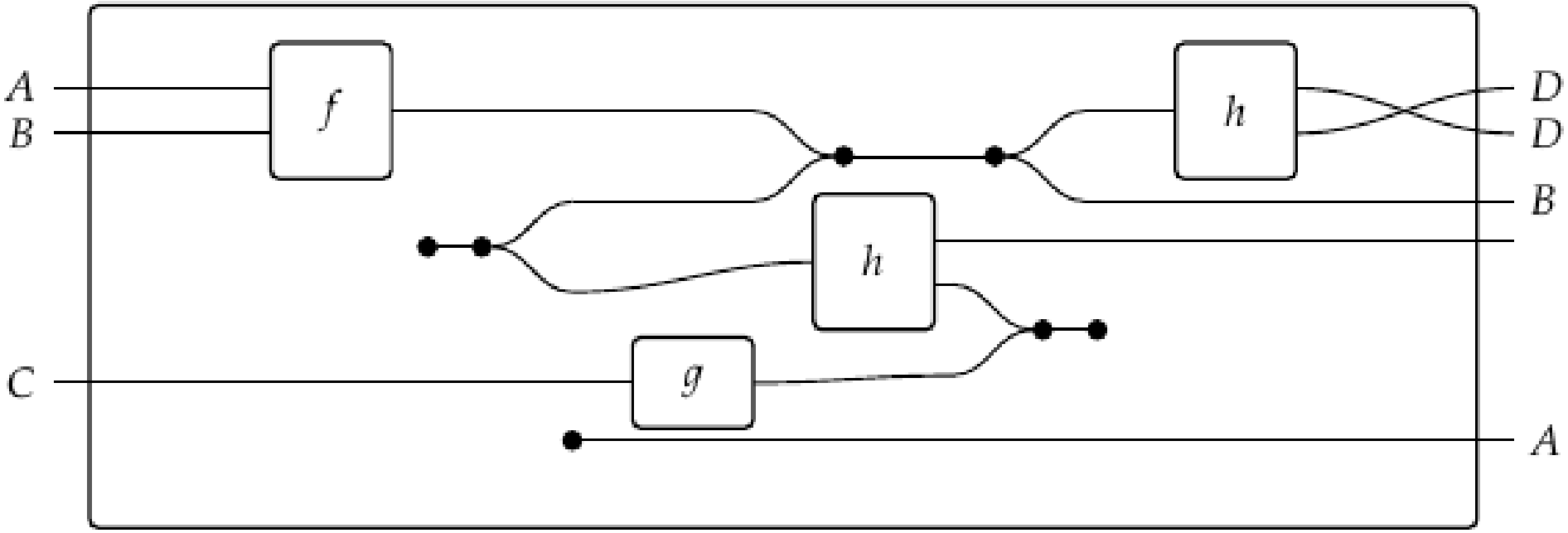
```
instance Buzz = eval qBuzz NASA
```

Q (2)

Row	Objective	Rocket
0	**Manned lunar landing**	**Saturn V**
1	**Extravehicular activity**	**Gemini-Titan II**

A Universal Language

- Logic is the language of true and false.
- Geometry is the language of shape and size.
- Calculus is the languge of motion and change.
- Category theory is the language of coherency and composition.



References

There's lots more to learn!

- Categorical Query Language: `http://www.categoricaldata.net/`
- *Seven Sketches in Compositionality* by Brendan Fong and David Spivak
- *What is Applied Category Theory?* by Tai-Danae Bradley
- *Category Theory for Programmers* by Bartosz Milewski
- *Conceptual Mathematics* by Stephen Schanuel and William Lawvere

CQL code.

```
/* run cql with jdbc connector
 *  java -cp "cql.jar:  " catdata.ide.IDE
 */
// connect CQL to an external database, PostgreSQL, for instance.
options
        jdbc_default_string = "jdbc:postgresql:postgres?user=postgres&password=docker"
        always_reload = true
// Define types in the database.
typeside Ty = literal {
        java_types
                String = "java.lang.String"
                Bool = "java.lang.Boolean"
        java_constants
                String = "return input[0]"
                Bool = "return java.lang.Boolean.parseBoolean(input[0])"
        java_functions
        Matches : String, String -> Bool = "return input[0].matches(input[1])"
}
schema SpaceAgency = literal : Ty {
        entities
                Mission Program Rocket Astronaut
        foreign_keys
                program : Mission -> Program
                rocket : Mission -> Rocket
                pilot : Mission -> Astronaut
        attributes
                name : Program -> String
                objective : Program -> String
                name : Mission -> String
                name : Rocket -> String
                name : Astronaut -> String
}
instance NASA = literal : SpaceAgency {
```

```
        generators
                mercury gemini apollo : Program
                mercury6 gemini4 gemini12 apollo11 apollo13 : Mission
                atlas titan saturn_v : Rocket
                glenn white aldrin haise : Astronaut
        multi_equations
                name -> {mercury "Mercury", gemini "Gemini", apollo "Apollo"}
                objective -> {mercury "Manned Earth orbital flight",
                              gemini "Extravehicular activity",
                              apollo "Manned lunar landing"}
                program -> {mercury6 mercury, gemini4 gemini,
                                        gemini12 gemini, apollo11 apollo, apollo13 apollo}
                rocket -> {mercury6 atlas, gemini4 titan,
                                     gemini12 titan, apollo11 saturn_v, apollo13 saturn_v}
                pilot -> {mercury6 glenn, gemini4 white,
                                    gemini12 aldrin, apollo11 aldrin, apollo13 haise}
                name -> {mercury6 "Mercury 6", gemini4 "Gemini 4",
                                   gemini12 "Gemini 12", apollo11 "Apollo 11",
                                   apollo13 "Apollo 13"}
                name -> {atlas "Mercury-Atlas",
                                   titan "Gemini-Titan II", saturn_v "Saturn V"}
                name -> {glenn "John Glenn",
                                    white "Edward H White II", aldrin "Buzz Aldrin",
                                    haise "Fred Haise"}
}
query qAstros = simple : SpaceAgency {
        from
          a : Astronaut
        attributes
          name -> a.name
}
instance Astros = eval qAstros NASA
query qBuzz = simple : SpaceAgency {
        from
          m : Mission
```

```
        where
          Matches(m.pilot.name, "Buzz.*") = true
        attributes
          Rocket -> m.rocket.name
          Objective -> m.program.objective
}
instance Buzz = eval qBuzz NASA
// Uncomment the following if you wish to export an instance to an external database.
// command Export = export_jdbc_instance NASA "" "nasa"
```