Visual Logic

as a universal interface for knowledge

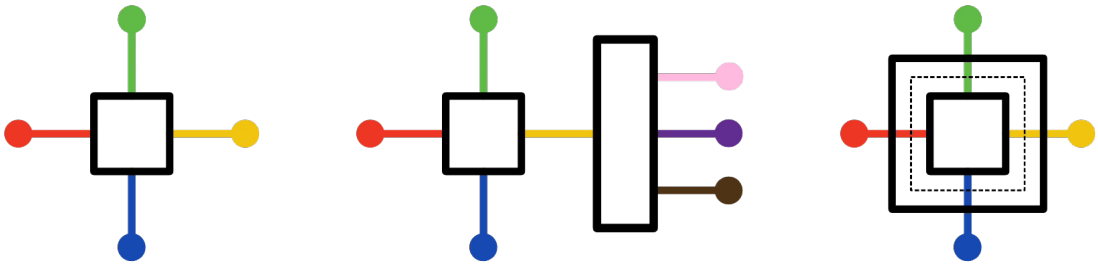
[CB Wells, 2024]

Introduction

Logic can empower us all, once it is simple to grasp and express and share.

Every kind of thinking shares the same general form: every thought expresses some *relation* of some types of things. Both in exploring nature, and creating society, we determine the world by forming **connections**.

Yet connections can be **visualized** in a simple way, as "nodes". We make thoughts by *composing* relations, and we reason from thought to thought by *inferring* from relation to relation; this can be visualized as "nesting" one node inside another.



relation composition inference

In this innocent idea lies great possibility: we can make a **universal interface for knowledge** — an infinite canvas for humanity — simply by seeing and holding thoughts as connections. The idea is to empower all people to explore and create.

The growth plan is to leverage the immense present demand for data science: most structured information is stored in *relational databases*, in which a table is a relation of its column types; so, visual logic is a **simple and complete data tool**. All data activity — exploration, creation, management, and integration — can be expressed through simple interactions with tables as nodes.

Upon completing a PhD in Mathematics [2], I have formed the company **Holdea**. Beginning open-source development of the explore data tool, to build the user base; seeking initial **funding**, expert **coding**, and **business/legal** support. Let's meet!

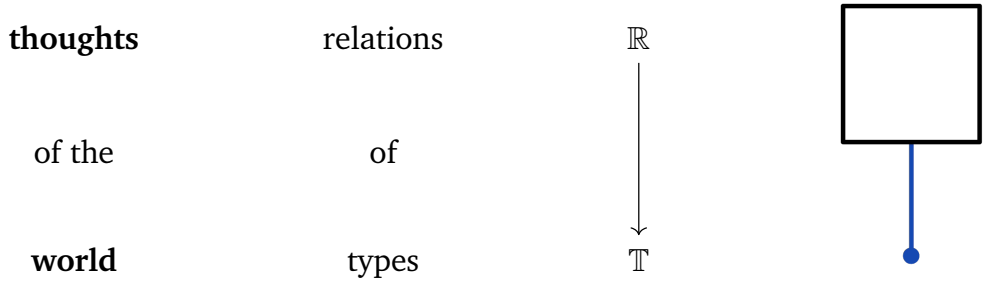
CB Wells
cb@holdea.co

Visual Logic

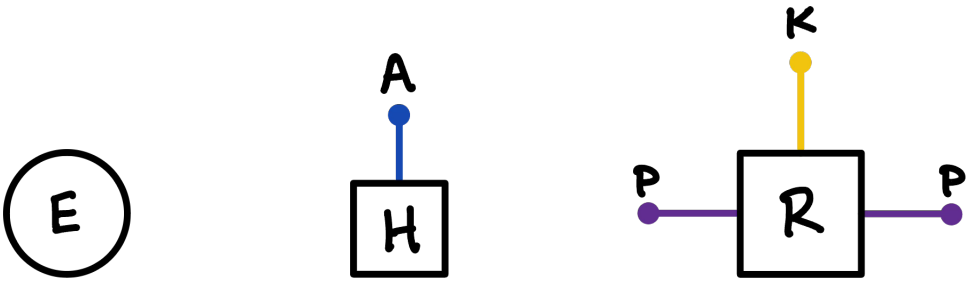
A **logic** is a two-layer structure:

1. a "world": a category $\mathbb{T}$ of *types* and *processes*, and
2. a "way of thinking": a category $\mathbb{R}$ of *relations* and *inferences*.

As thoughts are *of* the world, $\mathbb{R}$ depends on $\mathbb{T}$; this is known as a **fibered category**.



We can visualize a type as a **wire**, and a relation as a **node** that connects wires. Nodes can be any shape; wires can go any direction, and be colored by datatype.

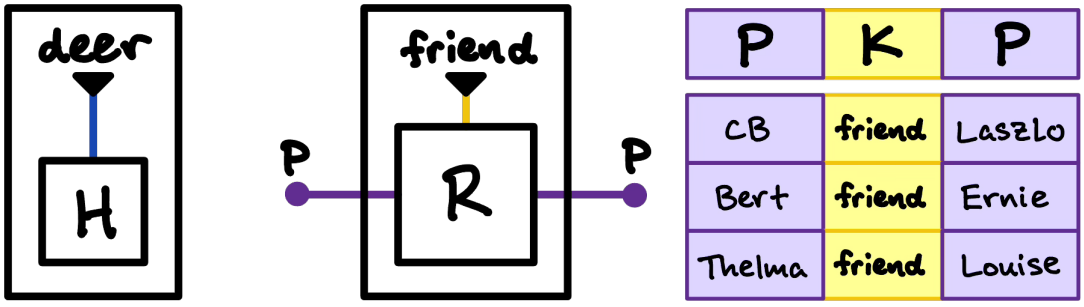


Earth is round

Animal a
is an Herbivore

Person p
has Relationship k with
Person q

Propositions like "Earth is round" are just true or false; other relations are *tables*, where a row is a related bunch of things. For only one type of thing, we call a relation a *predicate* or *property*. We can explore a table by *plugging in* specific things of each type, to determine whether they are related.



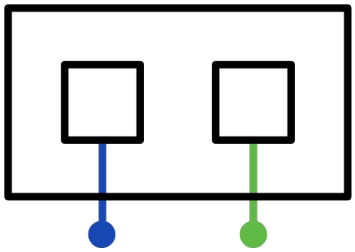
a deer is an Herbivore

table of friends

In this way, we can see thoughts in their simple shared form as *connections*, and we can build thoughts by composing connections, out of only four basic operations.

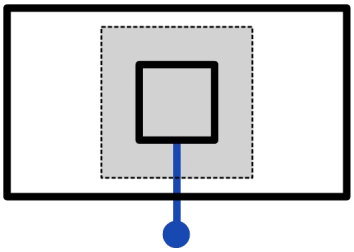
First, *conjunction* and *negation*: "and" is seen as placing side-by-side; while "not" is seen as alternating between two opposite background colors.

and



`P[a] and Q[b]`

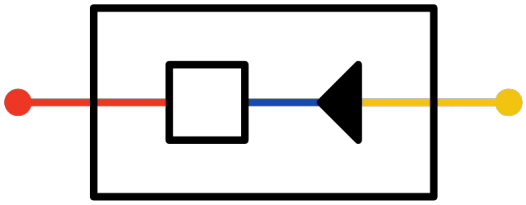
not



`not P[a]`

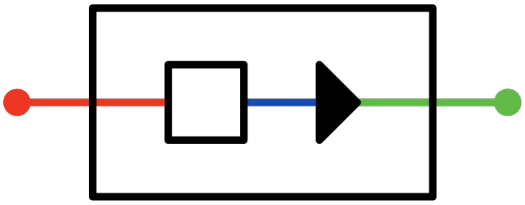
Secondly, processes can *act* on relations: we can *substitute* or "plug" a process into a relation, and we can form the *existential* or "sum" of a relation by a process.

plug



`R[a, b=f(y)]`

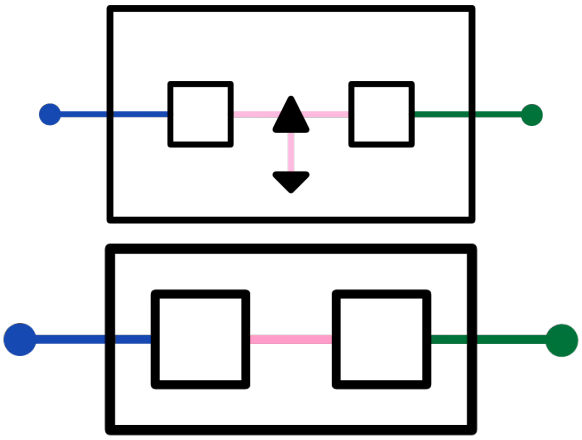
some



`some b : R[a, b] and [g(b)=c]`

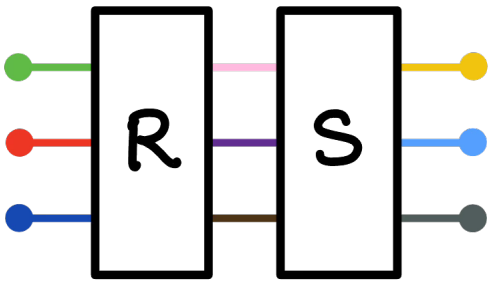
(Often, we use the process `g` that *deletes* the thing, `g(b)=*`, drawn as whitespace; so in this case, we form the relation `[some b : R[a, b]]` on things of type `A`.)

In particular, we *compose* relations by "and, plug, some": conjoin the relations, match the middle variable, and then sum. As the basic way we build connections, we can visualize composition simply as *connecting wires*.



`RS[a, c] = some b : R[a, b] and S[b, c]`

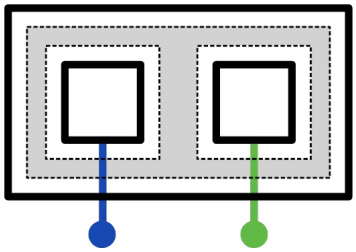
This is far more intuitive and expressive than in the Standard Query Language: all the millions of lines of "select-join" code can be left to the computers.




```
SELECT
R.C1,R.C2,R.C3,
S.C4,S.C5,S.C6
FROM R JOIN S
ON  R.C4=S.C1
AND R.C5=S.C2
AND R.C6=S.C3
```

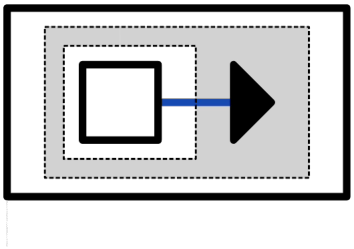
Relations and composition, made from "and" and "some", form the part of logic known as *regular logic*. By combining with negation, we form the *opposite half* of logic: disjunction "or", and universal "all". [1]

P[a] or Q[b]



not[not[P[a]] and not[Q[b]]]

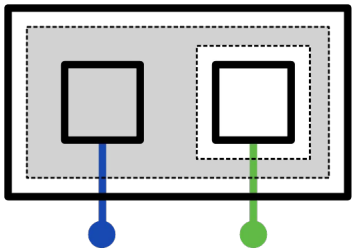
all b : R[a, b]



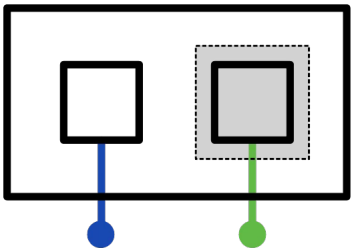
not[some b : not[R[a, b]]]

The other two basic operations are *implication* and *difference*.

if P[a] then Q[b]



not[P[a] and not Q[b]]

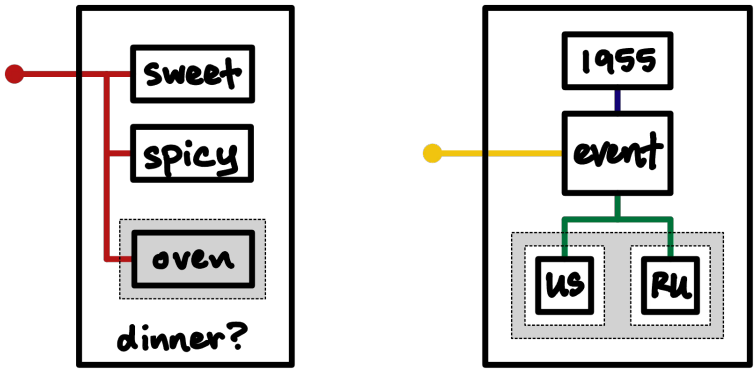


P[a] and not Q[b]

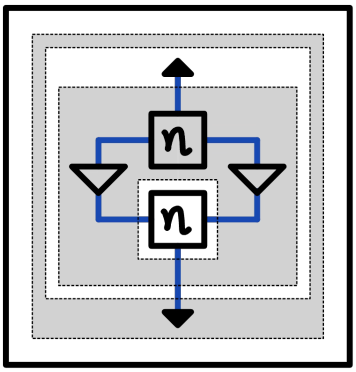
So, we can express all of **first-order logic** from just "and, not, plug, some".

This provides a lot; but **higher-order logic** includes our ability to see a *thought* as itself being a type of *thing* which we can think about. In databases, **aggregations** input tables rather than rows. This ability can be visualized as well (see appendix), providing complete expressiveness for thinking in all its forms.

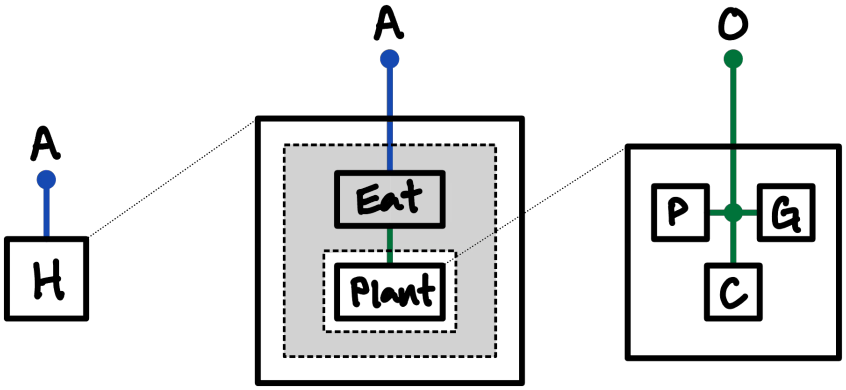
All our thoughts, in every sphere of life, we form out of these simple operations. They can be as simple as “what’s for dinner?” or as complex as world history;



or a basic notion, like *continuity* — read from outer to inner: for all distances e, there is some distance d so that if inputs are d-near, then outputs are e-near.



For us to grasp complex relations, we naturally view concepts in *layers of detail*. To learn the concept of Herbivore, for example, we can *expand* the herbivore node, to display its definition. Then we could ask “what is a Plant?” and expand *that* node — the visual logic interface can provide limitless exploration of knowledge.



a is an Herbivore

for all Organisms o
if a Eats o
then o is a Plant

o Photosynthesizes
and... etc.

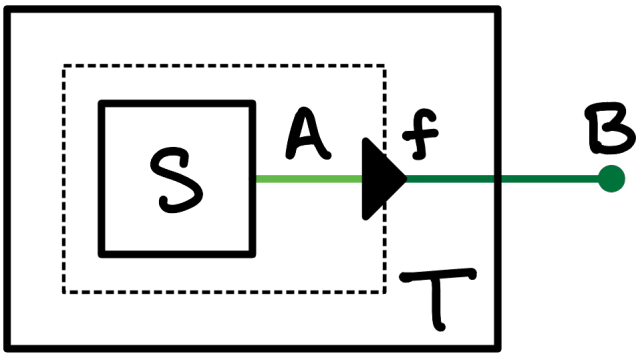
In computation, this is known as **modularity**. Any thought, no matter how complex, can be expanded to its full definition, or collapsed to a single node and built upon.

So far, this presents visual logic as a **query language**. Yet *exploring* knowledge is only one aspect; we also need to *create* and *update* knowledge.

Updates and Migrations

Things change — as a **process** $\mathtt{f} : \mathtt{A} \to \mathtt{B}$ transforms a type of thing into another, we can make **inferences** from A-thoughts to B-thoughts; this can be visualized as *nesting* a source node S into a target node T, along the process f from A to B.

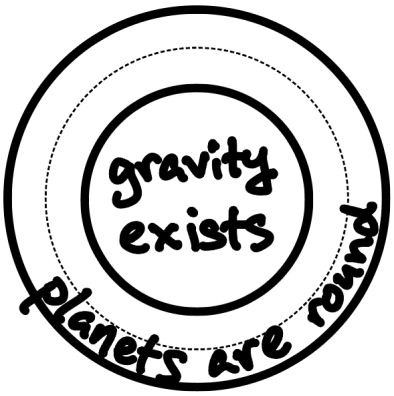
Below is the rule: if we know S of $\mathtt{a} : \mathtt{A}$, then we know T of $\mathtt{f}(\mathtt{a}) : \mathtt{B}$.



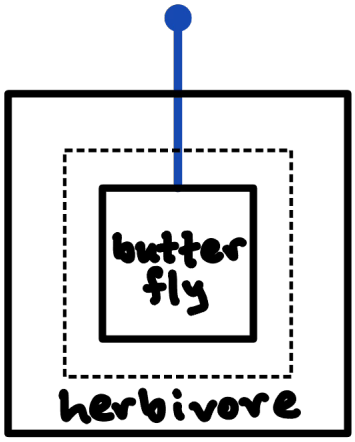
$$S[a] \vdash T[f(a)]$$

In a database, f is some **program** which modifies data, and the inference is an **update rule**: if a is a row in table S, then $\mathtt{f}(\mathtt{a})$ is a row in table T.

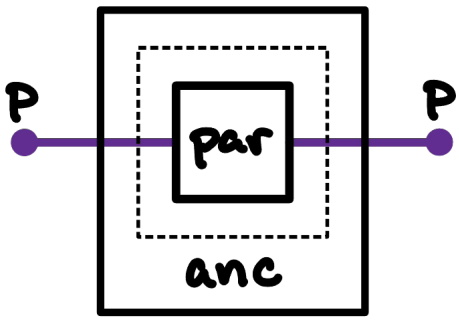
Here are examples where the things stay the same, as in $\mathtt{f}(\mathtt{a}) = \mathtt{a}$.



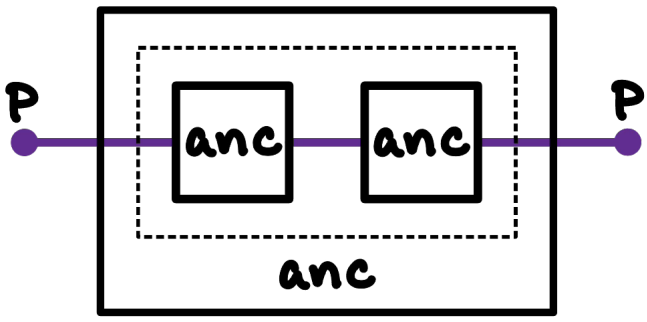
if gravity exists,
then planets are round.



if Animal a is a butterfly,
then a is an Herbivore.

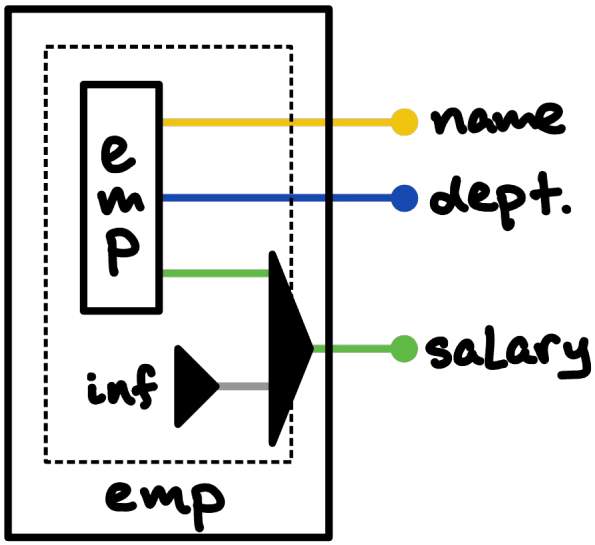


parent ⊢ ancestor

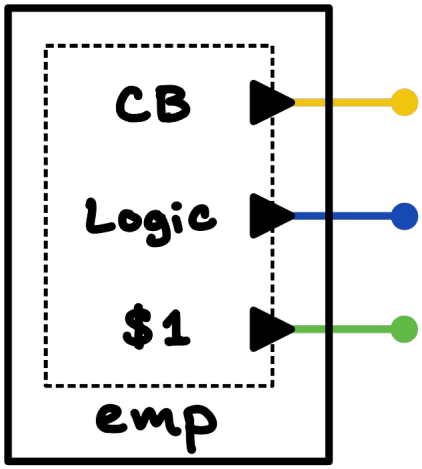


ancestor of an ancestor ⊢ ancestor

For a business example, suppose a company should adjust wages for inflation. This can be automated by a simple rule from and to the employee table (at left).



$\texttt{emp}[\texttt{n},\texttt{d},\texttt{s}] \;\vdash\; \texttt{emp}[\texttt{n},\texttt{d},\texttt{s}\times(1+\texttt{i})]$



$* \vdash \texttt{emp}[\texttt{CB},\texttt{Logic},\$1]$

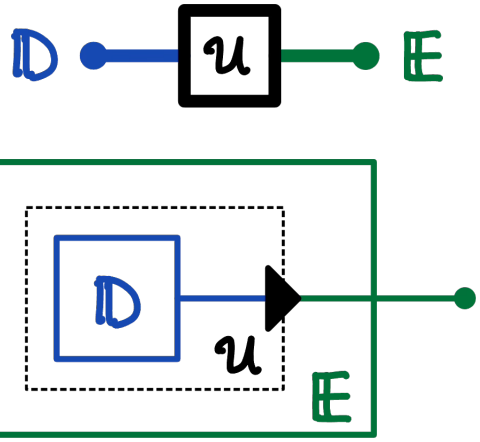
The source can have any conditions for when to execute, such as $\texttt{date}()=\texttt{Jan 1}$.

In particular, *creating* new knowledge is making rules with no preconditions. Filling a row b in a table T (at right) is asserting "if true, then we know T of $\texttt{b}()$".

So, a database **schema** (a *logic*, or an *ontology*) is a system of tables and rules; and an **instance** (the actual *data* of a database) is a system of rules "from true".

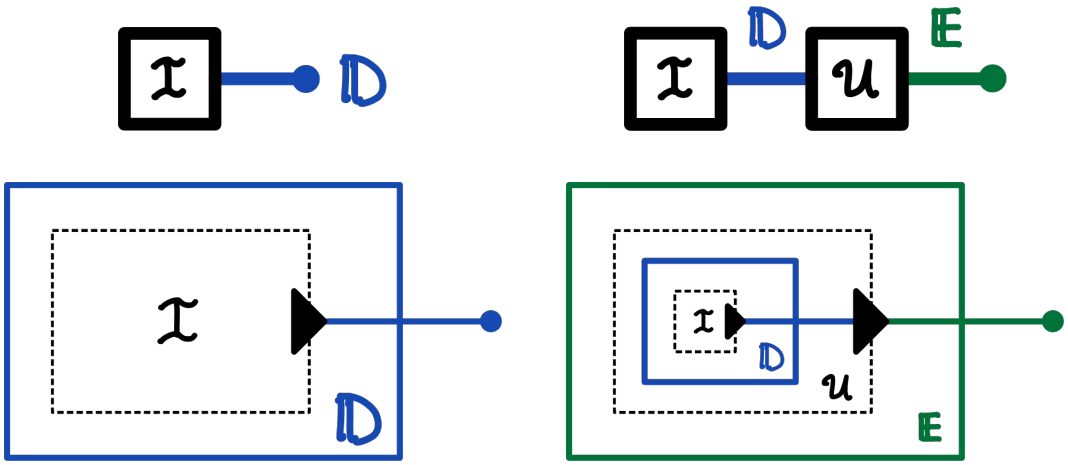
In full generality, we can form any system of rules between any pair of logics — this is known a **migration** between databases, and could be called a *transference* between logics. To see this higher view, we can "zoom out" to *the logic of logics*. (This is the main new idea of the author's doctoral thesis [2].)

In "metalogic", a type is a whole logic (database), and a relation $\mathcal{U}$ from $\mathbb{D}$ to $\mathbb{E}$ is a *transference* or **transfer**: a system of processes (programs) from types $\mathsf{D} : \mathbb{D}$ to types $\mathsf{E} : \mathbb{E}$, and inferences (updates) from relations (tables) in $\mathbb{D}$ to those in $\mathbb{E}$.



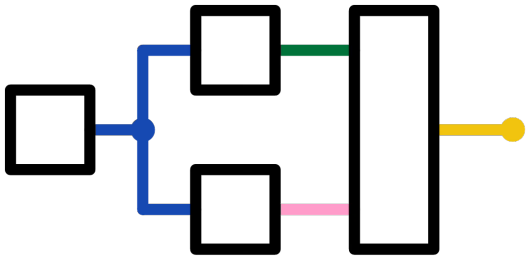
This is a migration between databases, or an "extract, transform, load" workflow; and this includes any **database update**: the logic $\mathbb{E}$ could be the same as $\mathbb{D}$, or it could be $\mathbb{D}$ plus some new tables, which are populated by the rules of $\mathcal{U}$.

Below is an instance $\mathcal{I}$ of schema $\mathbb{D}$, and the application of an update.

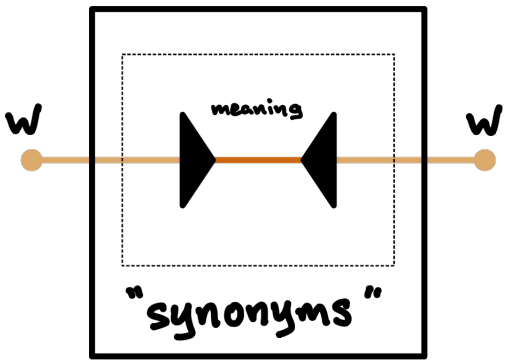


In an updated instance, each datum has a whole **history**, which can be queried!

From the bird's-eye view, we can easily see and manage *branching* updates, and we can *recombine* those branches; this encompasses all kinds of **data integration**.



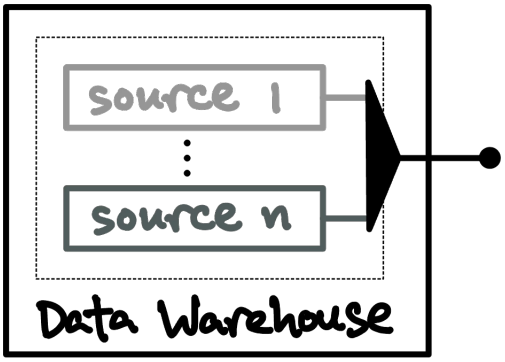
We create all kinds of logics, of all kinds of worlds; and we share and grow by relating our logics. So transfers are *everywhere*: **learning**, updating one's own logic, or **communicating**, one person updating their logic in response to someone;



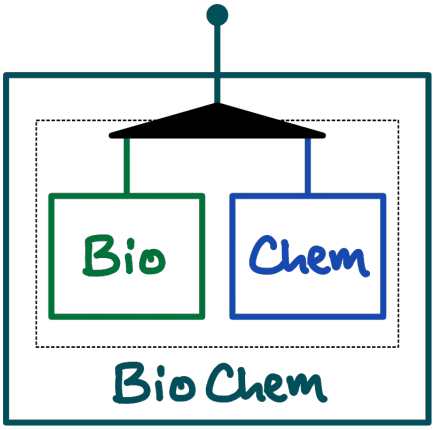
learning a new word

meeting a new friend

or **integrating** our knowledge — vital for most human endeavors.



Any company managing multiple data sources needs systematic migrations.



Some whole sciences can be understood as developing integrations.

Of course, such large-scale unification requires *interoperability*; so the interface must support many languages and systems. This is a large yet well-defined task.

As may be clear by now, the uses of a logical interface cannot be summarized, because logic underlies every aspect of human life. You can organize your thoughts, learn a new subject, run a business, design a social contract, imagine a new world. Visual logic is a canvas, meant to empower and connect humanity.

The development of the interface is now begun, by the new company Holdea. This project is open to everyone — feel free to message cb@holdea.co. Thank you.

References

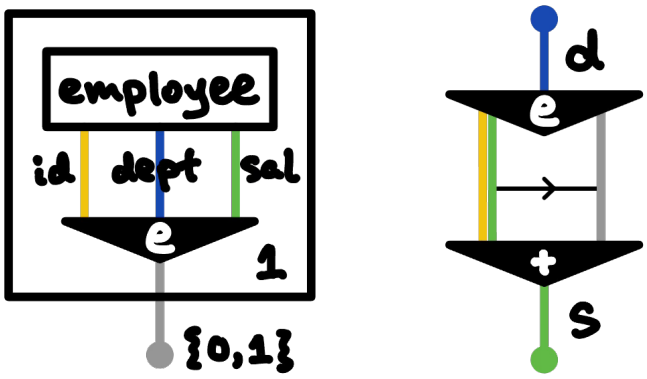
[1] Filippo Bonchi, Alessandro Di Giorgio, Nathan Haydon, and Pawel Sobocinski. Diagrammatic algebra of first order logic, 2024. Available at https://arxiv.org/abs/2401.07055.

[2] Christian Williams (now CB Wells). *The Metalanguage of Category Theory*. PhD thesis, University of California, Riverside, 2023. Available at https://escholarship.org/uc/item/84j4z67h.

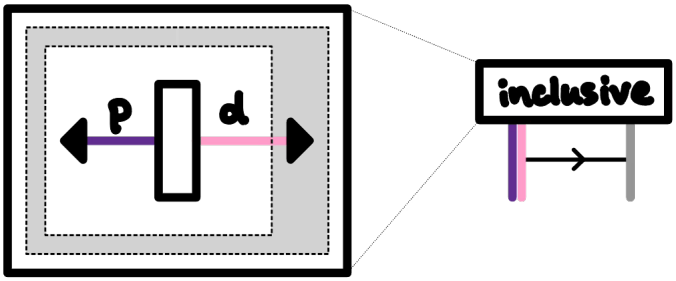
Appendix

In data science, an *aggregate function* inputs a whole table rather than one row, such as summing many values, or returning the highest value, or counting rows.

Converting a relation into a type can be visualized: we can see a table as its truth function, and "bend wires backward" to form a family of tables. For example, below is the query: "for each department, show the total salary of its employees".



This forms the basis of *higher-order logic*, the complete language which encompasses both "thinking" and "thinking *about* thinking". It goes well beyond aggregation — as a brief (and oversimplified) example, we could define a table of People to be Inclusive if all Demographics are represented.



Just by leaving the table name *blank*, we define a property of all tables of that type. We can reason about our systems, to ensure they are safe, effective, democratic, etc.