Shared Logic Interface
(software demonstration)

C. B. Wells

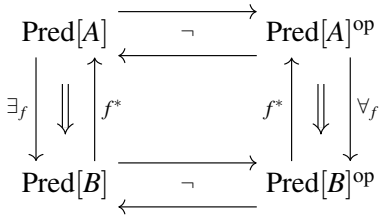
We present a logic interface, which enables users to manage data and execute code (Rust + datalog) using string diagrams. It is a desktop app, which works offline, and it can be hosted on any server. The repository and a test server will be made public in June.

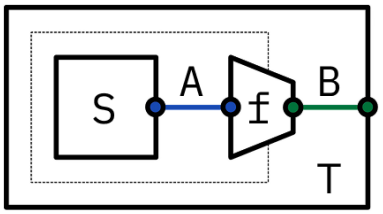
1 Overview

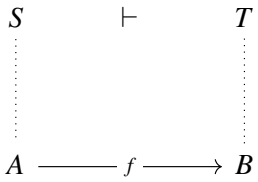
The visualization of logic is based on a simple idea: every statement is a relation (a.k.a. predicate), which can be seen as a connection; and we build ideas simply by composing these connections.

A relation is a node with ports, and an equality is a wire between ports; see Spivak and Fong [3]. Negation alternates the canvas between two opposite colors; see Sobocinski et. al. [2].

Yet first-order logic as a union of two halves $(\vdash, \exists, \wedge)$ and $(\dashv, \forall, \vee)$ can be also understood directly as a self-dual regular logic: a regular fibered preorder [4, 4.4] with negation as a fiberwise duality. (left)







Entailment can then be visualized as the containment of a premise relation within a conclusion relation. This is because a monoidal fibered category forms a monoidal double category (its collage), which is a special case of the string diagrams introduced in the author's thesis [5]. [1]

Rules can analyze data, send messages, or move resources – so a logic interface is a tool for both knowledge and action. Logic underlies all human endeavor, both in exploring nature and creating society. The goal is to empower people to understand, design, and cooperate in the systems of a shared world.

2 Architecture

The app runs on a PC, or a server. The two work the same; but the former persists data with SQLite, and the latter with Postgres – and the latter has many users, hence permissions. The personal logic works offline, and the app can be connected to any number of shared logics; and these all populate the same UI.

The frontend is ReactFlow, and the backend is Rust: SeaORM provides unified db management, and Ascent embeds datalog in Rust; this is the engine which runs user programs.

3 Schema

The minimal data of a logic consists of just a few sets – of types, functions, relations, and rules. Yet in practice, we adjoin extra data: for instance, we associate a name to a parameter of a function or relation. To utilize a logic as a platform that is visual, evolving, shared – this involves a rich metadata schema.

[1] However, an explicit ring to divide inner and outer is necessary for full correctness, and this was neglected in the thesis; but it is now rectified in the interface and forthcoming papers.

Submitted to:
ACT 2026

```
core  module, datatype, relation, column, function, rule, domain
exec  program, program_entity, compilation, execution
mgmt  action, transaction, fact_relation, derivation
ntwk  peer, connection, transfer
org   user, role, ability, user_role, role_ability
ui    blob, description, visual_data
```

Every fact has three metadata columns: `id, creation_id`, and `deletion_id`. The id is a global unique identifier, which acts as a link; this enables nesting of facts. The creation_id links to an action, which belongs to a transaction, which has a user_id or an execution_id, and a time.

All history is preserved: when a fact is "updated", a new fact is created, and an action links the old deletion_id to the new creation_id. Even schema history is fully preserved – the only alterations to physical tables is the addition of columns, and reading facts from a certain time shows only the columns which were created and not yet "deleted". Thus we see not only the evolution of facts, but concepts.

Metadata provides many significant capabilities. For any set of ports in the canvas, we can generate all ways to connect those ports. For any relation, we can view all rules in which it occurs as premise. For any fact generated by a rule, we can view its derivation tree – e.g. to trace a bad conclusion to its source.

4 Visualization

There are many aspects; we mention only some. To navigate rules, a conclusion relation can be expanded into its premises. A coproduct is viewed as a cospan apex; this suffices for case statements and most uses. Since lists, bags, etc. are built from products and sums, all such type constructors are visualized.

The facts of a relation form a hypergraph; each fact is a hyperedge, a copy of the relation node whose ports are connected to data-value nodes. Force-directed layout distributes them in the canvas, providing automatic visualization of social networks, supply chains, etc.

Datatypes are equipped with finite domains, enabling function graphs and computation of preimages.

Function types will be supported [1], providing higher-order logic and metaprogramming.

5 Execution

A program is assembled from Rust functions and datalog rules, then compiled and executed in Wasm – either once, generating all results from the input, or continuously, so that new facts trigger live updates.

This extends to communication between servers: when logic B has a key to logic A, users of B can define rules with premise in A and conclusion in B – then as long as there is internet connection, such rules will transfer data in the same way as local rules. So, the interface will support a network of logics.

References

[1] J. Baez and M. Stay. Physics, topology, logic and computation: A rosetta stone. page 95–172, 2010. Available at `https://arxiv.org/abs/0903.0340`.

[2] 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`.

[3] Brendan Fong and David I Spivak. Graphical regular logic, 2019. Available at `https://arxiv.org/abs/1812.05765`.

[4] B. Jacobs. *Categorical Logic and Type Theory*. Elsevier, Amsterdam, 1998.

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