Red-Wine Categorical DB Bridge

Concrete repo-local cSQL pullbacks, pushouts, and truth witnesses

Simon Frost

Table of contents

Introduction 1
Setup 1
Inspect the study surface 2
Build the categorical DB bridge 2
Materialize the concrete truth witnesses 3
Compile the bridge into parity artifacts 3
Why this matters 4

Introduction

One of the main remaining v1 parity gaps was the concrete categorical database demo backed by repo-local sample data. That gap is now closed in Julia. This vignette walks through the red-wine cardio/resveratrol study using the actual `FunctorFlow_v1/sample_data/demo_repo_root` assets:

1. inspect the recovered study metadata;
2. build the categorical DB bridge;
3. materialize the exact pullback, soft pullback, and pushout; and
4. lower the bridge to semantic compiler artifacts.

Setup

```
using Pkg
Pkg.activate(joinpath(@__DIR__, ".."))

using FunctorFlow
```

```
sample_root = normpath(joinpath(@__DIR__, "..", "..", "..", "FunctorFlow_v1", "sample_data", "
isdir(sample_root) || error("Expected FunctorFlow_v1 sample data at $sample_root")
```

```
true
```

Inspect the study surface

The high-level study descriptor tells us which atlas pair is being glued and which SQL scripts define the exact/soft pullback and pushout surfaces.

```
study = describe_red_wine_csql_study(sample_root)

println("Study: ", study["name"])
println("Atlas A: ", study["atlas_a"])
println("Atlas B: ", study["atlas_b"])
println("Tables: ", study["tables"])
println("Focus terms: ", study["focus_terms"])
```

```
Study: red_wine_cardio_resveratrol
Atlas A: Dict{String, Any}("nodes" => 3, "name" => "atlas_cardio", "role" => "cardio", "top_hu
Atlas B: Dict{String, Any}("nodes" => 3, "name" => "atlas_resveratrol", "role" => "resveratrol
Tables: Dict("exact_pullback" => "pullback_edges", "soft_pullback" => "pullback_resv_soft", "p
Focus terms: ["resveratrol"]
```

Build the categorical DB bridge

`build_red_wine_csql_categorical_bridge` gives a Julia-native categorical object for the bridge itself: two atlas objects, a shared claim-key base, an exact pullback, a soft pullback, and a pushout.

```
bridge = build_red_wine_csql_categorical_bridge(sample_root)

println("Base object: ", bridge.base_object.name)
println("Atlas objects: ", (bridge.atlas_a_object.name, bridge.atlas_b_object.name))
println("Exact pullback output table: ", bridge.exact_pullback.output_table)
println("Soft pullback output table: ", bridge.soft_pullback.output_table)
println("Pushout output table: ", bridge.pushout.output_table)
```

```
Base object: RedWineClaimKeyBase
Atlas objects: (:RedWineCardioDBObject, :RedWineResveratrolDBObject)
Exact pullback output table: pullback_edges
Soft pullback output table: pullback_resv_soft
Pushout output table: pushout_edges
```

The bridge is intentionally Julian in shape, but it preserves the normalized semantic outputs of the Python v1 implementation.

Materialize the concrete truth witnesses

The materializer executes the recovered workflow against the local Parquet data and classifies the resulting pushout edges into practical cSQL truth values.

```
materialization = describe_red_wine_csql_materialization(sample_root; witness_limit=4)

println("Truth value counts: ", materialization["truth_value_counts"])
println("Table counts: ", materialization["table_counts"])
println("Witnesses:")
for witness in materialization["witnesses"]
    println("  ", witness)
end
```

```
Truth value counts: Dict("A_ONLY" => 2, "CONSENSUS" => 1, "B_ONLY" => 2)
Table counts: Dict("exact_pullback" => 1, "pushout" => 5, "soft_pullback" => 1)
Witnesses:
  Dict{String, Any}("truth_value" => "CONSENSUS", "support_lcms_a" => 1, "source" => "resverat
```

In this concrete sample the pushout contains a consensus witness linking resveratrol to blood pressure, plus atlas-specific witnesses that remain on only one branch of the pushout.

Compile the bridge into parity artifacts

The semantic compiler treats the categorical DB bridge and its universal constructions as first-class subjects.

```
plan = compile_plan(
    :RedWineBridgePlan,
    bridge.exact_pullback,
    bridge.soft_pullback,
    bridge.pushout,
    bridge,
)

println("Compiled bridge subjects:")
for artifact in plan.artifacts
    println("  ", (artifact.subject_name, artifact.subject_kind))
end
```

```
Compiled bridge subjects:
  (:RedWineExactPullback, :csql_pullback)
  (:RedWineSoftPullback, :csql_pullback)
  (:RedWinePushout, :csql_pushout)
  (:red_wine_cardio_resveratrol__categorical_db_bridge, :categorical_db_bridge)
```

Lowering to executable IR exposes the bridge-level operations directly.

```
ir = lower_plan_to_executable_ir(plan)

println("Bridge IR instructions:")
for instruction in ir.instructions
    println("  ", instruction.name, " ⇒ ", instruction.opcode)
end
```

```
Bridge IR instructions:
  RedWineExactPullback__csql_pullback => compose_csql_pullback
  RedWineSoftPullback__csql_pullback => compose_csql_pullback
  RedWinePushout__csql_pushout => compose_csql_pushout
  red_wine_cardio_resveratrol__categorical_db_bridge => declare_categorical_db_bridge
```

Why this matters

This is the first FunctorFlow.jl vignette that uses the actual v1 sample-data bridge rather than a synthetic semantic stand-in. That matters because it demonstrates all of the following in a single Julia-native workflow:

- repo-local atlas loading;
- exact and soft categorical pullbacks;
- pushout-based truth-value materialization; and
- semantic-compiler lowering over real, not invented, bridge outputs.