TCC Method Pullback

DID-vs-IV agreement and conflict surfaces over the Democritus causal-claims corpus

Simon Frost

Table of contents

Introduction

The TCC method-pullback demo lifts the Democritus causal-claims corpus into a categorical comparison between two method families:

- difference-in-differences (DID family), and
- instrumental variables (IV family).

The Julia port now materializes that pullback directly from the repo-local Parquet data and exposes both agreement counts and method-conflict witnesses.

Setup

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

using FunctorFlow
```

```
sample_root = normpath(joinpath(@__DIR__, "..", "..", "..", "FunctorFlow_v1", "sample_data", "
workspace_root = normpath(joinpath(@__DIR__, "..", "..", "..", "FunctorFlow_v1"))

isdir(sample_root) || error("Expected FunctorFlow_v1 sample data at $sample_root")
isdir(workspace_root) || error("Expected FunctorFlow_v1 workspace at $workspace_root")
```

```
true
```

Materialize the pullback summary

`describe_tcc_method_pullback` gives a normalized dictionary view that is stable across the Julia and Python implementations.

```
summary = describe_tcc_method_pullback(sample_root; top_k=6, workspace_root=workspace_root)

println("Compiled counts: ", summary["compiled_counts"])
println("Omega counts: ", summary["omega_counts"])
println("Top DID/IV pullback rows:")
for row in summary["did_iv_pullback"]
    println("  ", row)
end
```

```
Compiled counts: Dict("csql_edges" => 4, "csql_edge_support" => 7, "csql_nodes" => 6)
Omega counts: Dict("CONSENSUS_INC" => 2, "METHOD_CONFLICT" => 1)
Top DID/IV pullback rows:
  Dict{String, Any}("docs_iv" => 1, "docs_did" => 1, "mass_did" => 1.0, "sign" => "inc", "sour
  Dict{String, Any}("docs_iv" => 1, "docs_did" => 1, "mass_did" => 1.0, "sign" => "inc", "sour
```

The pullback rows expose claims that survive alignment across method families, while the omega counts summarize whether those aligned pairs land in consensus or method conflict.

Inspect explicit conflict witnesses

The same summary carries the method-conflict surface directly, which is useful for downstream review or adjudication.

```
println("Method conflicts:")
for row in summary["method_conflicts"]
    println("  ", row)
end
```

```
Method conflicts:
  Dict{String, Any}("sign" => "decrease", "source" => "minimum wage", "min_year" => 2018, "n_p
  Dict{String, Any}("sign" => "increase", "source" => "minimum wage", "min_year" => 2020, "n_p
```

Work with the typed Julia object

The typed `TCCMethodPullbackSummary` is useful when you want field access and semantic compilation rather than just a normalized report.

```
pullback = materialize_tcc_method_pullback(sample_root; top_k=6, workspace_root=workspace_root

println("Typed summary counts: ", pullback.compiled_counts)
println("First pullback witness: ", first(pullback.did_iv_pullback))
println("First conflict witness: ", first(pullback.method_conflicts))
```

```
Typed summary counts: [("csql_edge_support", 7), ("csql_edges", 4), ("csql_nodes", 6)]
First pullback witness: TCCMethodPullbackWitness("school spending", "inc", "test scores", 1, 1
First conflict witness: TCCMethodConflictWitness("minimum wage", "employment", "DID_FAMILY", "
```

Compile and execute the parity artifact

The semantic compiler lowers the pullback summary into a dedicated `tcc_method_pullback` artifact and a placeholder operation `materialize_tcc_pullback`.

```
artifact = compile_v1(pullback)
println("Artifact subject: ", (artifact.subject_name, artifact.subject_kind))
println("Artifact nodes: ", [(node.name, node.node_kind) for node in artifact.nodes])
```

```
Artifact subject: (:TCCMethodPullbackSummary, :tcc_method_pullback)
Artifact nodes: [(:TCCMethodPullbackSummary__node, :tcc_method_pullback)]
```

```
plan = compile_plan(:TCCMethodPullbackPlan, pullback)
ir = lower_plan_to_executable_ir(plan)
executed = execute_placeholder_ir(ir)

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

```
println("Execution trace:")
for line in executed.trace
    println("  ", line)
end
```

```
IR instructions:
  TCCMethodPullbackSummary__node => materialize_tcc_pullback
Execution trace:
  TCCMethodPullbackSummary__node: materialize_tcc_pullback inputs=() outputs=(:TCCMethodPullba
```

Why this matters

This vignette turns the TCC method-family comparison into a fully documented Julia-native workflow:

- the sample-data-backed pullback is materialized locally;
- agreement and conflict witnesses are visible as structured outputs; and
- the result lowers cleanly into the same parity-oriented compiler story as the rest of Functor-Flow v1.

That gives the Julia port a real worked example for empirical method comparison, not just symbolic categorical structure.