Democritus Assembly

Local causal graph gluing, repaired global sections, and regrounding

Simon Frost

Table of contents

Introduction

The basic `democritus_assembly_pipeline` block already exposes a sheaf-like local-to-global pattern: glue local claims into a global section, then map that global section back down to local fragments. This vignette adds the missing piece for a fuller CATAGI-style story: repair-aware assembly.

In the example below, local causal fragments come from three partially overlapping views of a labor-market story:

- a policy fragment,
- a household-demand fragment, and
- a labor-outcome fragment.

The assembly step glues them into a global causal graph and then infers repaired cross-fragment claims by taking a small transitive closure. The regrounding step pushes those repaired claims back into the local fragments whose focus vocabularies can express them.

Setup

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

using FunctorFlow
```

Build the repaired assembly example

```
example = build_democritus_assembly_example()

println("Local fragments:")
for (fragment, claims) in example[:local_claims]
    println("  ", fragment, " ⇒ ", sort(collect(claims)))
end

println("\nFragment focus vocabularies:")
for (fragment, focus) in example[:fragment_focus]
    println("  ", fragment, " ⇒ ", sort(collect(focus)))
end

println("\nOverlap cover:")
for (region, fragments) in example[:overlap_relation]
    println("  ", region, " ⇒ ", fragments)
end
```

```
Local fragments:
  policy => ["minimum wage -> earnings", "minimum wage -> labor costs"]
  labor => ["employment -> job quality", "labor costs -> employment"]
  household => ["demand -> employment", "earnings -> demand"]

Fragment focus vocabularies:
  policy => ["earnings", "employment", "labor costs", "minimum wage"]
  labor => ["employment", "job quality", "labor costs", "minimum wage"]
  household => ["demand", "earnings", "employment", "minimum wage"]

Overlap cover:
  policy_labor => [:policy, :labor]
  policy_household => [:policy, :household]
  labor_market => [:policy, :household, :labor]
```

This example uses a custom gluing reducer, `democritus_repair_reducer`, rather than a plain set union. The reducer first glues the local claims and then adds repaired claims such as `minimum wage → employment` that appear only after joining multiple local pieces.

Execute the assembly

```
result = execute_democritus_assembly_example(example)
summary = summarize_democritus_assembly_example(example)

println("Counts: ", summary["counts"])
println("Coherence / repair loss: ", summary["coherence_loss"])
```

```
Counts: Dict("repaired_claims" => 5, "global_claims" => 11, "seed_claims" => 6, "local_fragmen
Coherence / repair loss: 0.4
```

The assembled global section is keyed by overlap region. In this example the main region is `:labor_market`.

```
global_sections = result.values[example[:diagram].ports[:global_output].ref]

println("Global labor-market section:")
for claim in sort(collect(global_sections[:labor_market]))
    println("  ", claim)
end
```

```
Global labor-market section:
  demand -> employment
  demand -> job quality
  earnings -> demand
  earnings -> employment
  employment -> job quality
  labor costs -> employment
  labor costs -> job quality
  minimum wage -> demand
  minimum wage -> earnings
  minimum wage -> employment
  minimum wage -> labor costs
```

The repaired claims are exactly the claims that appear globally but were absent from the original local fragments.

```
println("Inferred global repairs:")
for claim in summary["global_inferred_claims"]
    println("  ", claim)
end
```

```
Inferred global repairs:
  demand -> job quality
  earnings -> employment
  labor costs -> job quality
  minimum wage -> demand
  minimum wage -> employment
```

Reground repairs back to local fragments

The regrounding map sends repaired global claims back into any local fragment whose focus vocabulary contains both endpoints of the claim. That turns the global section into a source of local repair proposals.

```
println("Fragment-level repairs:")
for (fragment, repairs) in sort(collect(summary["fragment_repairs"]); by=first)
    println("  ", fragment, " ⇒ ", repairs)
end
```

```
Fragment-level repairs:
  household => ["earnings -> employment", "minimum wage -> demand", "minimum wage -> earnings"
  labor => ["labor costs -> job quality", "minimum wage -> employment", "minimum wage -> labor
  policy => ["earnings -> employment", "labor costs -> employment", "minimum wage -> employmen
```

Notice that the repair loss is non-zero: the regrounded local fragments now contain repaired claims that were not present in the original local sections. That is exactly the signal we want if we interpret Democritus assembly as a local-to-global consistency-and-repair procedure.

Inspect the diagram itself

The example still compiles down to an ordinary FunctorFlow diagram, so it can be inspected, adapted, or replaced with learned morphisms later.

```
diagram = example[:diagram]

println("Diagram name: ", diagram.name)
println("Ports: ", collect(keys(diagram.ports)))
println("Operations: ", collect(keys(diagram.operations)))
println("Losses: ", collect(keys(diagram.losses)))
```

```
Diagram name: DemocritusCausalAssembly
Ports: [:input, :relation, :global_output, :regrounded_claims, :coherence_loss]
Operations: [:glue__assemble_global_section, :reground_repaired_claims]
Losses: [:section_repair_loss]
```

The key categorical pieces are:

1. a right Kan gluing operation that builds the global section;
2. a regrounding morphism back into local claim space; and
3. a section repair loss that measures how much local repair was needed.

Why this matters

This dedicated example pushes the Democritus block beyond a schematic macro:

- it uses a concrete local causal graph story;
- it exposes repaired claims that only appear after global assembly; and
- it shows how those repairs can be propagated back to the local fragments that need them.

That makes the block suitable as a template for multi-document causal synthesis, fragmentary world-model repair, and other local-to-global assembly workflows.