Reactive DCE: Distributed Graph Aggregation + Incremental Fixpoint

1 Overview

This document describes how Dead Code Elimination (DCE) can be implemented as a two-layer reactive system:

1. **Layer 1 (Reactive Aggregation)**: Combine file fragments into a global graph using reducers.
2. **Layer 2 (Incremental Fixpoint)**: Maintain the live set using the generic incremental fixpoint combinator from `incremental_fixpoint_notes.tex`.

DCE is an instance of incremental fixpoint where:

$$\mathsf{base} = \mathsf{roots}, \quad \mathsf{stepFwd}(u) = \{v \mid (u, v) \in \mathsf{edges}\}$$

2 Layer 1: Distributed Graph Aggregation

In a distributed system, the program graph is spread across files. Each file contributes a *fragment*:

$$f = (\mathsf{nodes}_f, \mathsf{roots}_f, \mathsf{edges}_f)$$

We aggregate fragments into a global graph using multiset union:

$$G = \bigoplus_i f_i = \left(\sum_i \mathsf{nodes}_i,\ \sum_i \mathsf{roots}_i,\ \sum_i \mathsf{edges}_i \right)$$

Reducer operations.

$$\begin{aligned} \iota &= (\emptyset, \emptyset, \emptyset) \\ G \oplus f &= (G.\mathsf{nodes} + f.\mathsf{nodes},\ G.\mathsf{roots} + f.\mathsf{roots},\ G.\mathsf{edges} + f.\mathsf{edges}) \\ G \ominus f &= (G.\mathsf{nodes} - f.\mathsf{nodes},\ G.\mathsf{roots} - f.\mathsf{roots},\ G.\mathsf{edges} - f.\mathsf{edges}) \end{aligned}$$

This reducer is well-formed: $(G \oplus f) \ominus f = G$ (multiset cancellation).

Deduplication. The incremental fixpoint operates on sets, not multisets. Before passing to Layer 2:

- $\mathsf{roots} = \{r \mid G.\mathsf{roots}(r) > 0\}$ (elements with positive count)
- $\mathsf{edges} = \{(u, v) \mid G.\mathsf{edges}(u, v) > 0\}$

3 Layer 2: Incremental Fixpoint

Given the aggregated graph G, the live set is defined as:

$$\mathsf{live} = \mathsf{lfp}(F) \quad \text{where} \quad F(S) = \mathsf{roots} \cup \{v \mid \exists u \in S.\, (u, v) \in \mathsf{edges}\}$$

This is exactly the pattern handled by the generic incremental fixpoint combinator. See `incremental_fixpoint_notes.tex` for:

- The expansion algorithm (BFS) for when roots/edges are added
- The contraction algorithm (well-founded cascade) for when roots/edges are removed
- Correctness proofs (formalized in Lean)
- Complexity analysis (delta-bounded)

4 Skip Service Architecture

A Skip service implementing reactive DCE:

1. **File collection**: `files : EagerCollection<FileId, Fragment>`
2. **Graph aggregation** (Layer 1): Use `reduce` with the fragment reducer to produce a single `GraphState`.
3. **Live set** (Layer 2): Use the managed fixpoint API (`SkipruntimeFixpoint`) with:
   - `base` = deduped roots from the aggregated graph
   - `step` = deduped edges from the aggregated graph
4. **Dead set**: `nodes` $\setminus$ `live`

When files change, Layer 1 updates the aggregated graph, and Layer 2 incrementally updates the live set using the fixpoint combinator's `applyDelta`.

5 References

- `incremental_fixpoint_notes.tex` — Generic incremental fixpoint theory and algorithms
- `IncrementalFixpoint.lean` — Formal correctness proofs
- `reduce.tex` — Reducer calculus for reactive aggregation