LatticeHashForest

(Work in Progress)

Anamitra Ghorui
<aghorui (at) cse.iitb.ac.in>

Prof. Uday Khedker
<uday (at) cse.iitb.ac.in>

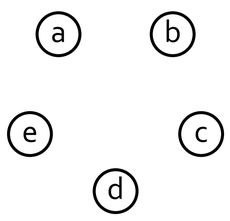
PLATO CSE IIT Bombay

[PLACID 2025]

Core Problem

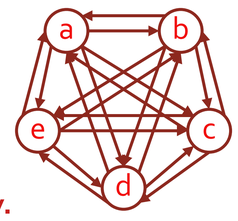
Usual Data Flow Analysis:

- Information Increases approximately linearly *(number of variables).*
  Eg: *Liveness Analysis, Constant Prop.*



Pointer/Points-to Analysis (PTA)

- **Information increases combinatorially.**
  (proportional to number of variables, statements, contexts and fields)
- **Consequently, Algorithmic cost multiplies combinatorially.**
- **Not Scalable, at least when done naively.**



- Main share of expense comes from the **mundane operations** that are performed, such as **comparisons and checking sets for equality**.
- Points-to, and information from other data-flow analyses is often **sparse and redundant** at a significant number of program points.
- Unions, intersections and other similar set operations may often result in the **same value redundantly computed**.

- A significant reason for this, *besides* theoretical ones, is the **naive** manner in which these operations and data structures are implemented.
- The overwhelming costs of operations and the sparseness of actual data is not taken into account in the **data structure** perspective of the problem.

Analysis Component (in LFCPA)	Total Cost Share (Inclusive)
Liveness Analysis in LFCPA	57.16%
Operand Comparison	**45.33%**
Points-to Analysis in LFCPA	28%
D.F.V. Equality Comparisons	**23%**

(Data from one of our PTA benchmarks. Total runtime: 16 mins.)

Prior Work and Present Situation

Our goal at IITB: Precise & scalable flow, context and field sensitive pointer analysis on **a million lines of code in tens of minutes.**

- **Early Work: Compromises for Scalability:**
  Andersen's Analysis, Steensgaard's Analysis
  - **Remove Flow Sensitivity.**
  - **Remove Context Sensitivity.**
  - **Imprecise, approximated results.**
- **Current: Some inroads made, but still no clear answer for doing a precise analysis, while still being scalable.**
  No large scale whole-program precise, flow and context sensitive points-to analysis done as of yet.

Work towards Precise PTA In IIT Bombay:

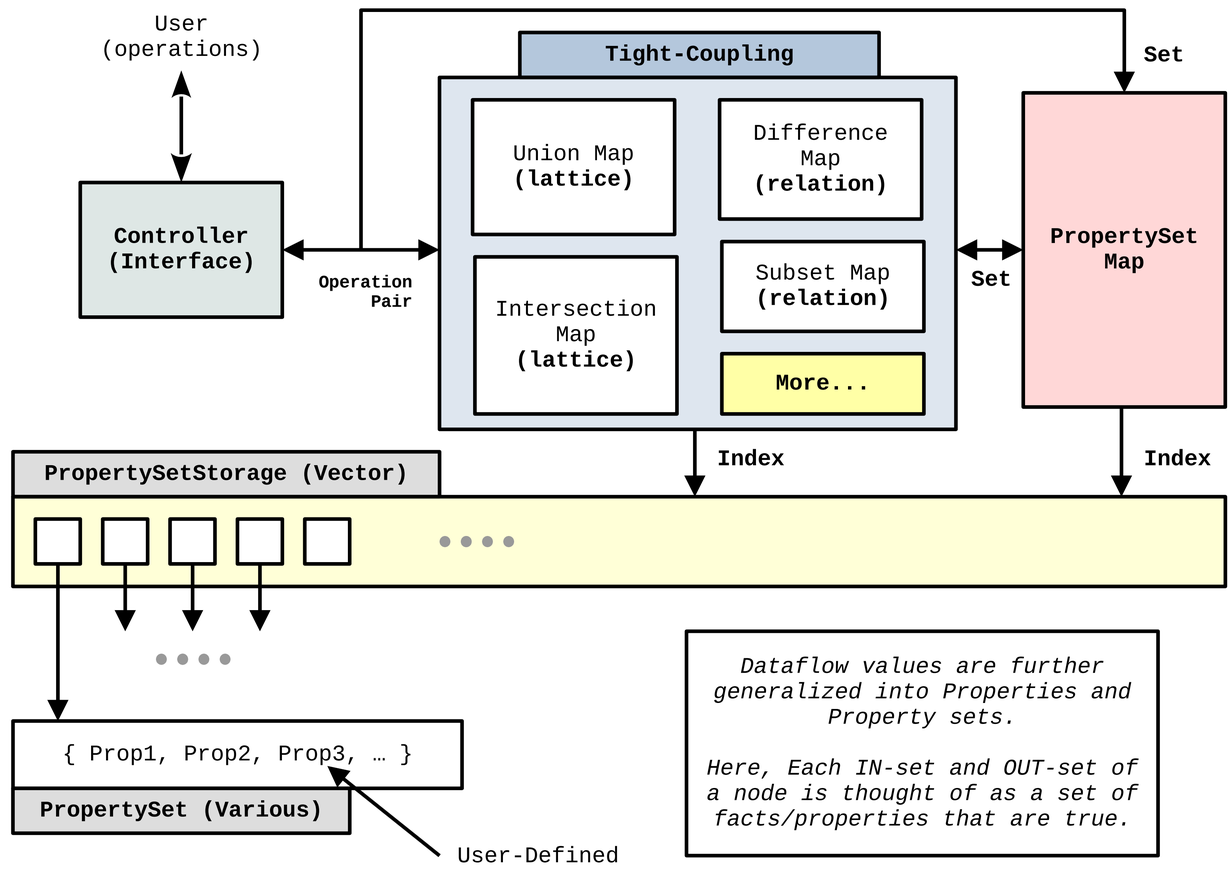
LFCPA | ByLFCPA | Sparse LFCPA | Gen. PTG

SLIM: Simplified LLVM IR Modelling

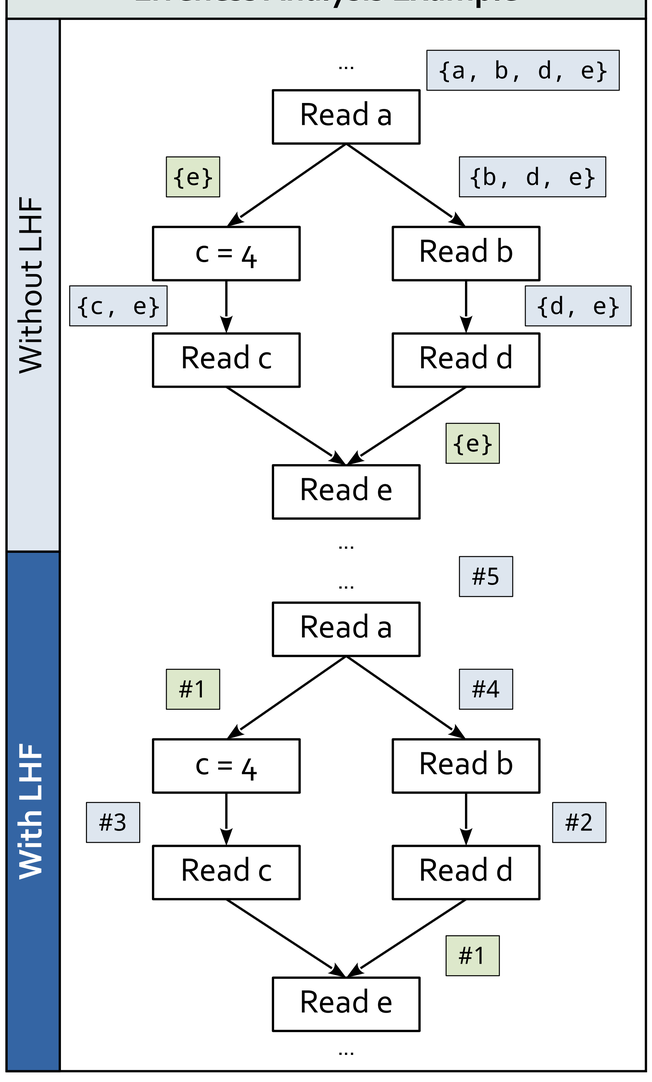
LatticeHashForest (LHF)

A Step Towards the Solution

- **Why not capture and memoise the results of all of these operations, and consequently the dataflow values that are propagating?**
- Store unique sets in a **vector**, hash them, and use **integer indexes** for each unique set. The sets are made **immutable**.
- For a given operation between two sets, perform the operation, and store **both** the operation and the result in hash tables, as well as **peripheral facts (like subset relations)**.
  [Index1, Index2] → **ResultIndex**
- This all results in a very **generic data structure for caching operations and redundant data** that can likely be used for many problem domains.
- Current implementation is in C++.



Liveness Analysis Example



Immutability and Deduplication give us such avenues for optimization.

Advantages and Second-Order Benefits

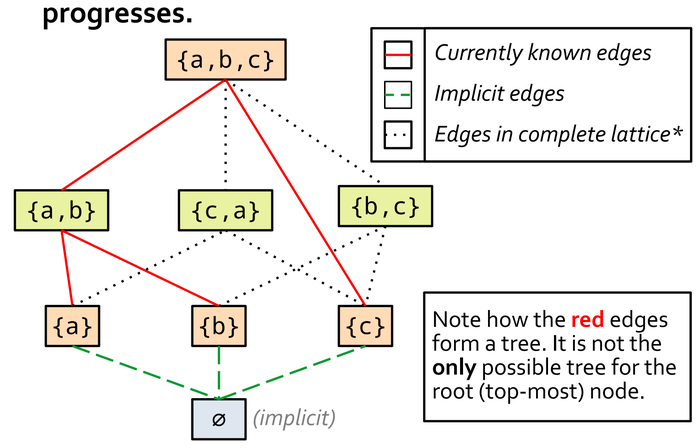
- **Operation Simplification:** Lot of operations reduce to **integer comparisons**.
- **Deduplication**: Cutting out redundant data.
- **Immutability**: Simplifies our management of Property Sets, and the choice of data structure: we can get away simply with **vectors that are sorted**.
- **Cross-Operation Information Sharing**: Facts obtained from one operation (like subset relations) can be used to make other operations faster.
- **Optimistic Insertion through Parallelization**: Since data is immutable, we can insert data that is more likely to be accessed **predictively** using a **worker thread** → a byproduct of immutability.
- **Analysis Pre-Caching**: We can use a cheap Pre-Analysis to **pre-cache possible results in the actual analysis**.

Work to be Done and Ideas to Explore

- **Integration into an actual analysis and subsequent evaluation**: currently the mechanism has been tested with random data, and **initial results seem optimistic**.
  - Evaluation metrics:
    - Hits based on Empty Sets, Subsets, Cached results
    - Cold misses, Operation misses, Time taken
- Explore parallelization, further generalization of the mechanism, statistical inferences, data structures…
- **Lazy-insertion** of Property Sets.
- Eventually, try to make **whole-program points-to analysis efficient.**

Why "LatticeHashForest"?

- As a consequence of the way we are constructing new sets from existing sets, **each** set in our storage vector is essentially a *collection* of all the **non-unique derivation trees of operations, forming a forest**.
- Common operations like Union or Intersection, form a special type of partial order with these sets known as a **lattice**.
- Merging all of these non-unique derivation trees gives us a **partial, incremental construction of a lattice for these operations as the analysis progresses.**



*Trivial edges have not been shown. See Hasse Diagrams.

Similar Work

Barbar, Mohamad, and Yulei Sui. "Hash Consed Points-To Sets." International Static Analysis Symposium. Cham: Springer International Publishing, 2021.