Cauliflower: A Solver-Generator for Context-Free Language Reachability

Nicholas Hollingum Bernhard Scholz

The University of Sydney

LPAR-21: 8th-12th May, 2017

What Is CFL-R?

A Datalog Fragment

- Formulated originally by Yannakakis [**?**]
- A subclass of Datalog:
  - Binary relations (EDB & IDB)
  - Chain rules:

$$H(v_0, v_k) \rightarrow B_1(v_0, v_1), B_2(v_1, v_2), \ldots, B_k(v_{k-1}, v_k).$$

A Datalog Fragment

- Formulated originally by Yannakakis [**?**]
- A subclass of Datalog:
  - Binary relations (EDB & IDB)
  - Chain rules:

$$H(v_0, v_k) \rightarrow B_1(v_0, v_1), B_2(v_1, v_2), \ldots, B_k(v_{k-1}, v_k).$$

- Rephrased as a graph problem:
  - Binary predicates $\Leftrightarrow$ labelled graph edges
  - Chain rules $\Leftrightarrow$ context-free grammar

$$H \rightarrow B_1 B_2 \ldots B_k$$

- A generalisation of two well-known computational problems

Recognition +

- A generalisation of two well-known computational problems

- Given a context-free language
- Determine if a string is a member of the language
- CYK (cubic time)
- Valiant's algorithm [**?**]

Recognition + Reachability

- A generalisation of two well-known computational problems

- Given a context-free language
- Determine if a string is a member of the language
- CYK (cubic time)
- Valiant's algorithm [**?**]

- Given a directed graph
- Determine if there exists a path between two vertices
- Only care about one path
- Transitive closure [**?**]

Recognition + Reachability

- A generalisation of two well-known computational problems

- Given a context-free language
- Determine if a string is a member of the language
- CYK (cubic time)
- Valiant's algorithm [**?**]

- Given a directed graph
- Determine if there exists a path between two vertices
- Only care about one path
- Transitive closure [**?**]

- By coincidence, both are equivalent to matrix multiplication

CFL-R Applications

- Points-to analysis [**?**, **?**, **?**, **?**, **?**]
- Dataflow analysis [**?**, **?**]
- Shape analysis [**?**]
- Constant propagation [**?**]
- Inter-procedural slicing [**?**]
- Some logic fragments [**?**]
- Set Constraints [**?**, **?**]
- Security verification [**?**, **?**]
- Control-flow analysis [**?**]

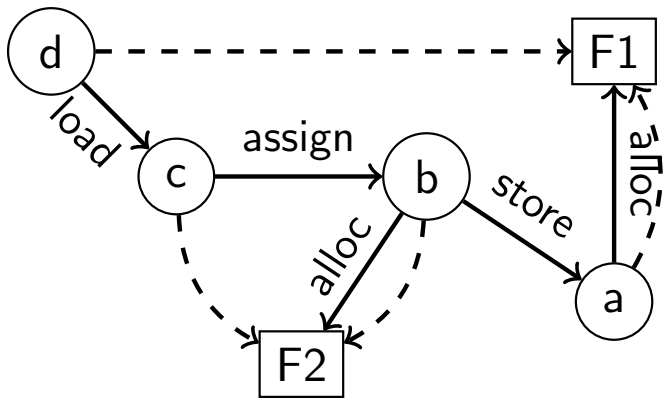
Application: Points-To

```
a = new F1();
b = new F2();
d = c.f;
b.f = a;
c = b;
```

- Which program variables refer to which (abstract) heap locations at runtime

Application: Points-To

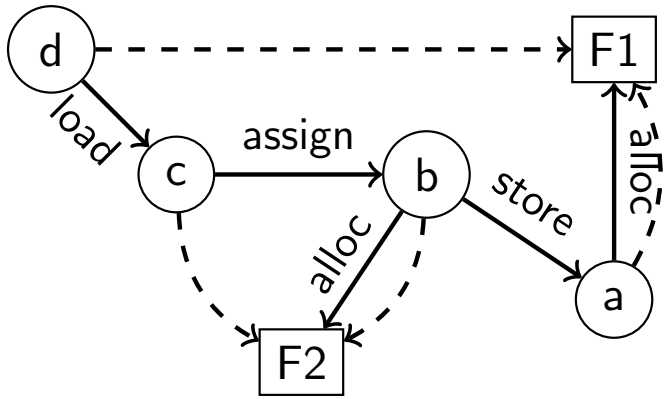
```
a = new F1();
b = new F2();
d = c.f;
b.f = a;
c = b;
```



- Which program variables refer to which (abstract) heap locations at runtime

Application: Points-To

```
a = new F1();
b = new F2();
d = c.f;
b.f = a;
c = b;
```



$pt \rightarrow alloc$

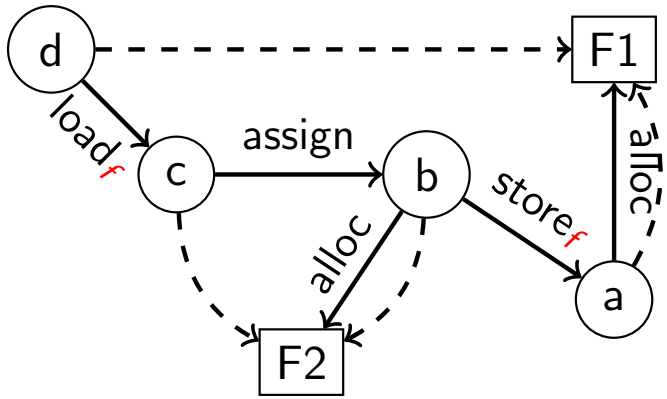
$pt \rightarrow assign\ pt$

$pt \rightarrow load \quad pt\ \overline{pt}\ store \quad pt$

- Which program variables refer to which (abstract) heap locations at runtime

Application: Points-To

```
a = new F1();
b = new F2();
d = c.f;
b.f = a;
c = b;
```



$pt \rightarrow alloc$

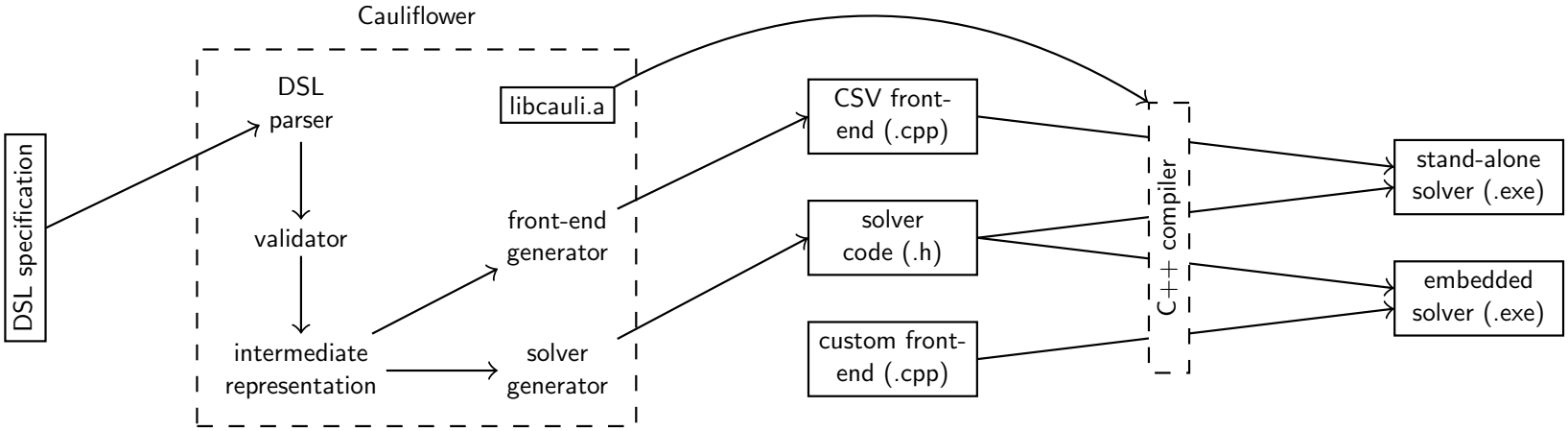
$pt \rightarrow assign\ pt$

$pt \rightarrow load_f\ pt\ \overline{pt}\ store_f\ pt$

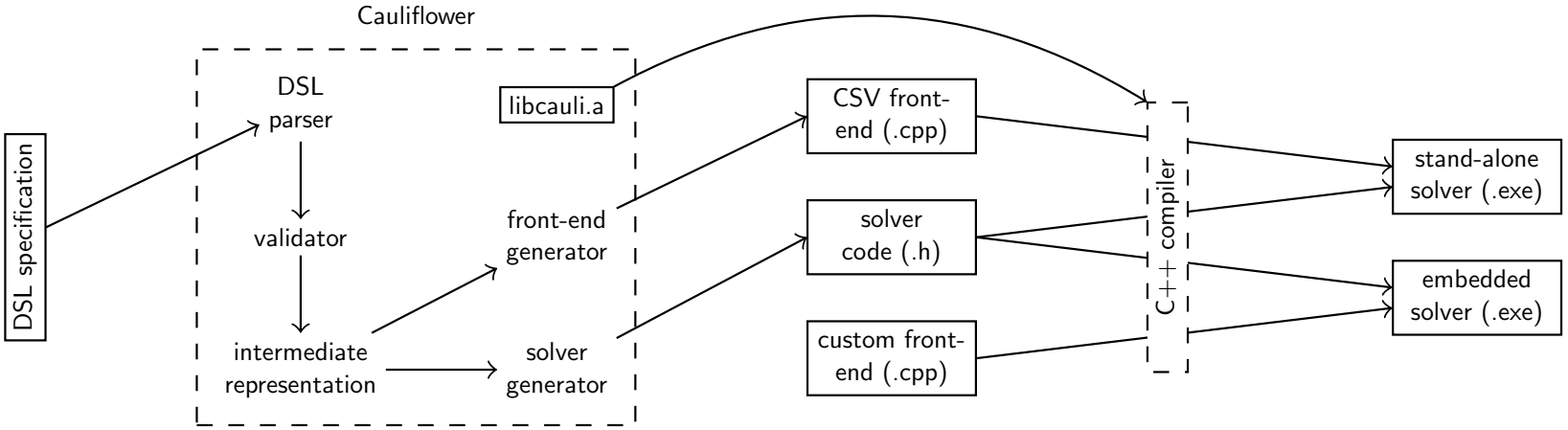
- Which program variables refer to which (abstract) heap locations at runtime
- Field sensitivity, improve precision by treating object fields distinctly

Improving CFL-R

Cauliflower

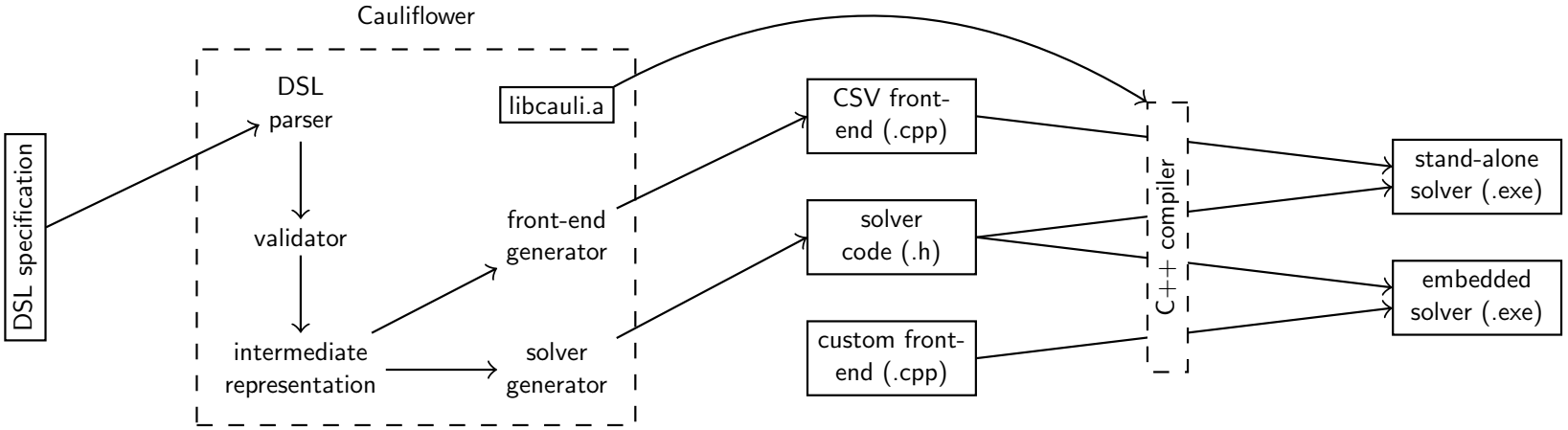


Cauliflower



- Solver generator
  - Typical use-case: one grammar many graphs
  - Problems encoded in a Domain-Specific language
  - Specialised solvers for the given problem
  - Static code generator (C++)
    - Avoids dynamic execution planning
    - Leverage c++ compiler optimisations

Cauliflower



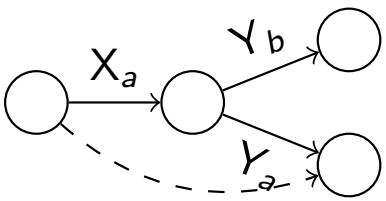
- Solver generator
  - Typical use-case: one grammar many graphs
  - Problems encoded in a Domain-Specific language
  - Specialised solvers for the given problem
  - Static code generator (C++)
    - Avoids dynamic execution planning
    - Leverage c++ compiler optimisations
- Optimiser?
  - Soon^{TM}

Enhanced Semantics

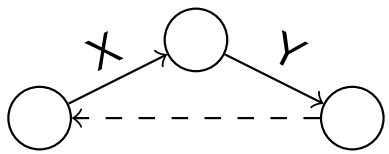
- In the literature, CFL-R alone is not enough
  - Points-to: $pt \rightarrow load_f \ pt \ \overline{pt} \ store_f \ pt$

Enhanced Semantics

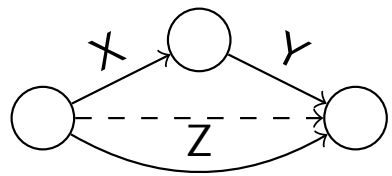
- In the literature, CFL-R alone is not enough
  - Points-to: $pt \rightarrow load_f \ pt \ \overline{pt} \ store_f \ pt$
- Templates for rules, reverse paths, branches, disconnection



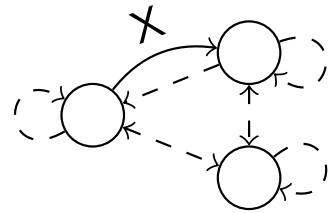
X[f], Y[f]



-(X, Y)



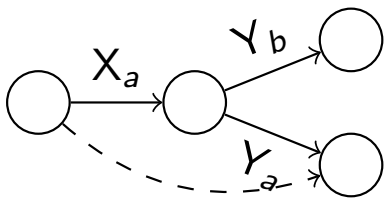
(x,y) & Z



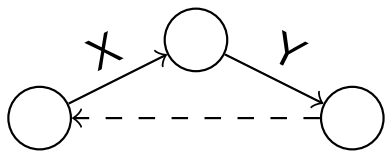
!X

Enhanced Semantics

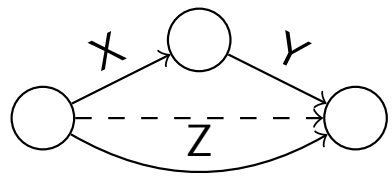
- In the literature, CFL-R alone is not enough
  - Points-to: $pt \rightarrow load_f \ pt \ \overline{pt} \ store_f \ pt$
- Templates for rules, reverse paths, branches, disconnection



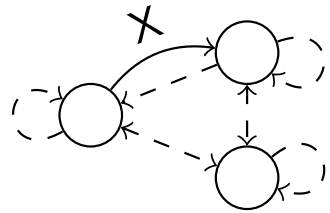
X[f], Y[f]



-(X, Y)



(x,y) & Z



!X

- Capture more problems in CFL-R
- Develop a DSL for CFL-R specifications
  - Rapid prototyping

CFL-R Domain-Specific Language

- Type declarations
  - Semantic correctness
  - Performance - partitioning

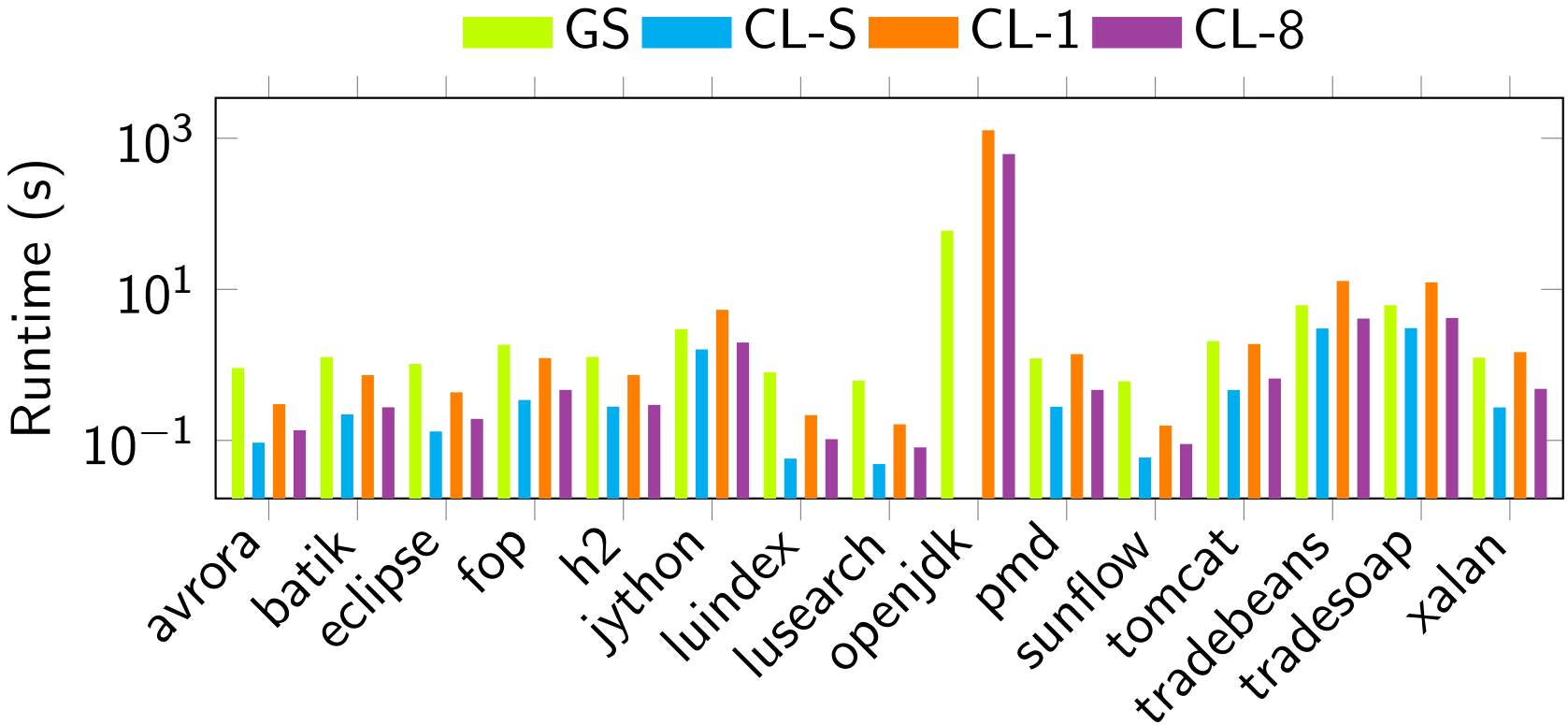
```
Alloc <- v . h;
Assign <- v . v;
Cast[t] <- v . v;
Load[f] <- v . v;
Store[f] <- v . v;
Bridge <- v . v;
VPT <- v . h;
LVPT[f] <- v . h;
SVPT[f] <- v . h;
```

- Rule declarations
  - BNF syntax
  - Enhanced semantics

```
Assign -> Cast[type];
VPT -> Alloc;
VPT -> -Assign, VPT;
VPT -> Bridge, VPT;
LVPT[f] -> -Load[f], VPT;
SVPT[f] -> Store[f], VPT;
Bridge -> LVPT[f],
          -SVPT[f];
```

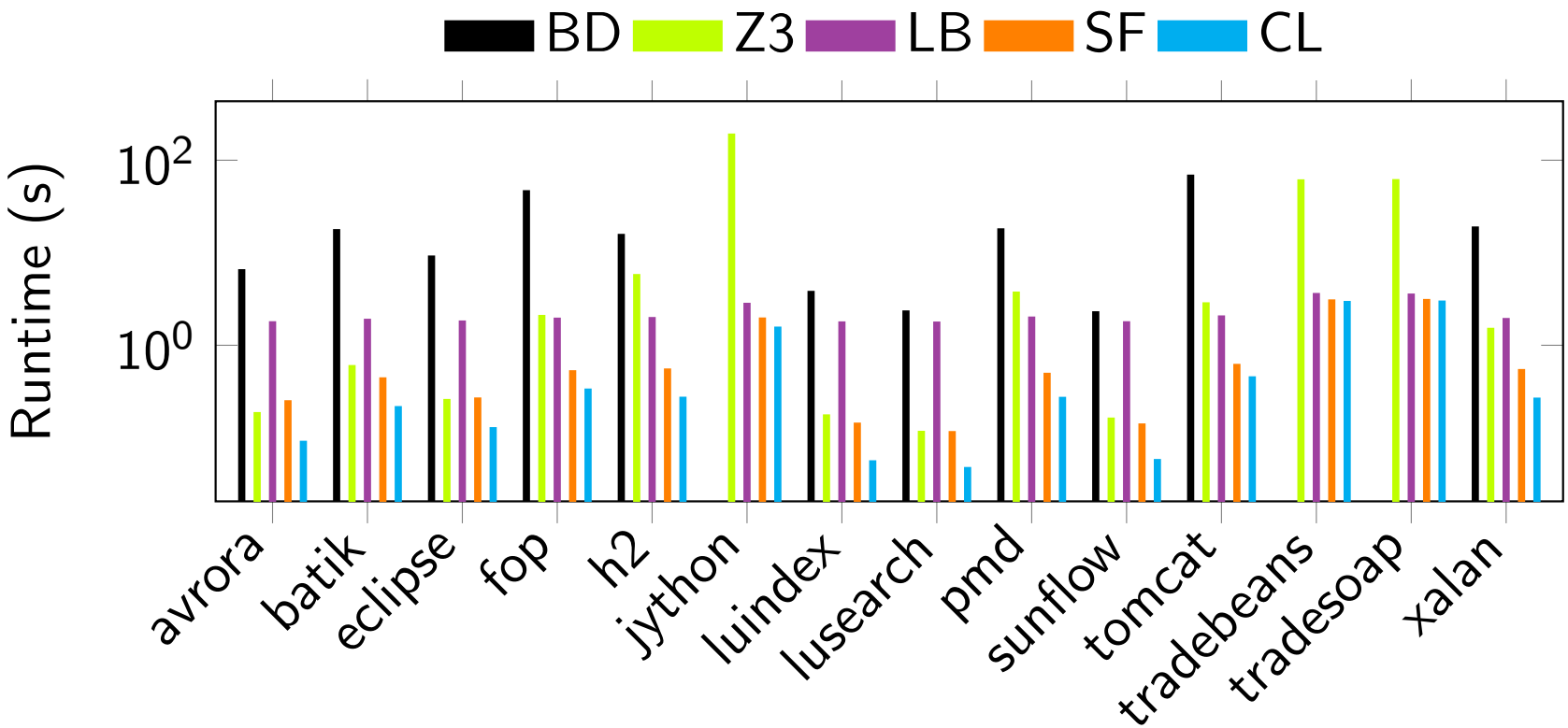
Experimental Validation

Comparison vs Gigascale



- Gigascale, high performance field-sensitive points-to analysis
- 1 hour timeout
- Cauliflower is more efficient on small cases
- Cauliflower solves large instance, but not as efficiently

Comparison vs Datalog



- Points to analysis - 1 hour timeout
- Datalog exhibits overheads/inefficiencies Cauliflower doesn't

Live Demo

Summary

Cauliflower

- CFL-R
  - Generalisation of Recognition and Reachability
  - Ubiquitous in program analysis
  - Notably points-to analysis
- Cauliflower
  - Solver generator
  - Parallel
  - Enhanced semantics
    - Fields, Reversal, Branching, Disconnection
  - DSL for rapid prototyping

References I


Bastani, O., Anand, S., and Aiken, A. (2015).
Specification inference using context-free language reachability.
In *Proceedings of the 42nd Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, pages 553–566. ACM.


Dolev, D., Even, S., and Karp, R. (1982).
On the security of ping-pong protocols.
Information and Control, 55(13):57 – 68.


Kodumal, J. and Aiken, A. (2004).
The set constraint/cfl reachability connection in practice.
In *Proceedings of the ACM SIGPLAN 2004 Conference on Programming Language Design and Implementation*, PLDI '04, pages 207–218, New York, NY, USA. ACM.

References II


Lu, Y., Shang, L., Xie, X., and Xue, J. (2013).
An incremental points-to analysis with cfl-reachability.
In Jhala, R. and De Bosschere, K., editors, *Compiler Construction*, volume 7791 of *Lecture Notes in Computer Science*, pages 61–81. Springer Berlin Heidelberg.


Melski, D. and Reps, T. (2000).
Interconvertibility of a class of set constraints and context-free-language reachability.
Theoretical Computer Science, 248(12):29 – 98. PEPM'97.


Nuutila, E. (1995).
Efficient transitive closure computation in large digraphs.
PhD thesis, PhD thesis, Helsinki University of Technology, 1995. Acta Polytechnica Scandinavica, Mathematics and Computing in Engineering Series.

References III


Reps, T. (1998).
Program analysis via graph reachability.
Information and Software Technology, 40(11-12):701–726.


Reps, T., Horwitz, S., and Sagiv, M. (1995).
Precise interprocedural dataflow analysis via graph reachability.

In *Proceedings of the 22Nd ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, POPL '95, pages 49–61, New York, NY, USA. ACM.


Sagiv, M., Reps, T., and Horwitz, S. (1995).
Precise interprocedural dataflow analysis with applications to constant propagation.
In Mosses, P., Nielsen, M., and Schwartzbach, M., editors, *TAPSOFT '95: Theory and Practice of Software Development*,

References IV

volume 915 of *Lecture Notes in Computer Science*, pages 651–665. Springer Berlin Heidelberg.

Sridharan, M., Gopan, D., Shan, L., and Bodík, R. (2005).
Demand-driven points-to analysis for java.
In *Proceedings of the 20th Annual ACM SIGPLAN Conference on Object-oriented Programming, Systems, Languages, and Applications*, OOPSLA '05, pages 59–76, New York, NY, USA. ACM.

Valiant, L. G. (1975).
General context-free recognition in less than cubic time.
Journal of Computer and System Sciences, 10(2):308 – 315.

Vardoulakis, D. and Shivers, O. (2010).
Cfa2: A context-free approach to control-flow analysis.
In Gordon, A., editor, *Programming Languages and Systems*, volume 6012 of *Lecture Notes in Computer Science*, pages 570–589. Springer Berlin Heidelberg.

References V


Yan, D., Xu, G., and Rountev, A. (2011).
Demand-driven context-sensitive alias analysis for java.
In *Proceedings of the 2011 International Symposium on Software Testing and Analysis*, ISSTA '11, pages 155–165, New York, NY, USA. ACM.


Yannakakis, M. (1990).
Graph-theoretic methods in database theory.
In *Proceedings of the Ninth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems*, PODS '90, pages 230–242, New York, NY, USA. ACM.


Zheng, X. and Rugina, R. (2008).
Demand-driven alias analysis for c.
SIGPLAN Not., 43(1):197–208.