The Flix Language

Magnus Madsen, **Ming-Ho Yee**, Ondřej Lhoták
University of Waterloo

March 4, 2016

What is Flix?

Flix is a declarative language for specifying and solving fixed-point computations on lattices.

Flix is inspired by Datalog, but supports lattices and functions.

What is Flix?

Flix is a declarative language for specifying and solving static program analyses.

Flix is inspired by Datalog, but supports lattices and functions.

What is Datalog?

Datalog is similar to the relational algebra, but is more expressive.

Every Datalog program terminates and has a least fixed point.

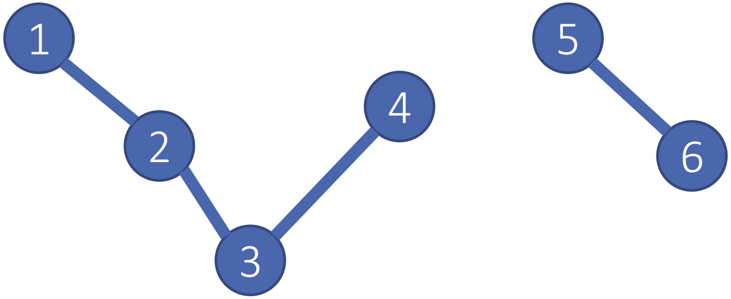
Example: Transitive Closure (1/2)

```
// Rules
Path(x, y) :- Edge(x, y).
Path(x, z) :- Path(x, y), Edge(y, z).
```

Head

Body

```
// Facts
Edge(1, 2).
Edge(2, 3).
Edge(3, 4).
Edge(5, 6).
```



Example: Transitive Closure (2/2)

```
Edge(1, 2). Edge(2, 3). Edge(3, 4). Edge(5, 6).

Path(x, y) :- Edge(x, y).
Path(x, z) :- Path(x, y), Edge(y, z).
```

Solution:

Example: Transitive Closure (2/2)

```
Edge(1, 2). Edge(2, 3). Edge(3, 4). Edge(5, 6).

Path(x, y) :- Edge(x, y).
Path(x, z) :- Path(x, y), Edge(y, z).
```

Solution:

```
Edge(1, 2), Edge(2, 3), Edge(3, 4), Edge(5, 6)
```

Example: Transitive Closure (2/2)

```
Edge(1, 2). Edge(2, 3). Edge(3, 4). Edge(5, 6).

Path(x, y) :- Edge(x, y).
Path(x, z) :- Path(x, y), Edge(y, z).
```

Solution:

```
Edge(1, 2), Edge(2, 3), Edge(3, 4), Edge(5, 6)
Path(1, 2), Path(2, 3), Path(3, 4), Path(5, 6)
```

Example: Transitive Closure (2/2)

```
Edge(1, 2). Edge(2, 3). Edge(3, 4). Edge(5, 6).

Path(x, y) :- Edge(x, y).
Path(x, z) :- Path(x, y), Edge(y, z).
```

Solution:

```
Edge(1, 2), Edge(2, 3), Edge(3, 4), Edge(5, 6)
Path(1, 2), Path(2, 3), Path(3, 4), Path(5, 6)
Path(1, 3), Path(2, 4)
```

Example: Transitive Closure (2/2)

```
Edge(1, 2). Edge(2, 3). Edge(3, 4). Edge(5, 6).

Path(x, y) :- Edge(x, y).
Path(x, z) :- Path(x, y), Edge(y, z).
```

Solution:

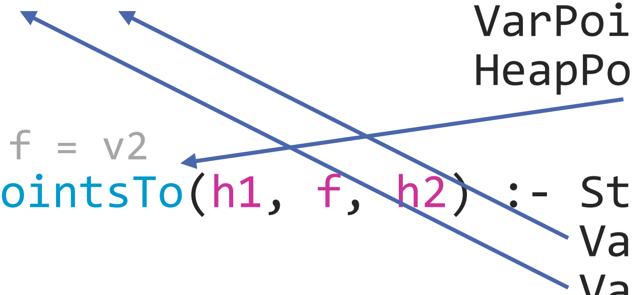
```
Edge(1, 2), Edge(2, 3), Edge(3, 4), Edge(5, 6)
Path(1, 2), Path(2, 3), Path(3, 4), Path(5, 6)
Path(1, 3), Path(2, 4)
Path(1, 4)
```

Example: Points-to Analysis

```
// v1 = new …
VarPointsTo(v1, h1) :- New(v1, h1).

// v1 = v2
VarPointsTo(v1, h2) :- Assign(v1, v2),
                       VarPointsTo(v2, h2).

// v1 = v2.f
VarPointsTo(v1, h2) :- Load(v1, v2, f),
                       VarPointsTo(v2, h1),
                       HeapPointsTo(h1, f, h2).

// v1.f = v2
HeapPointsTo(h1, f, h2) :- Store(v1, f, v2),
                           VarPointsTo(v1, h1),
                           VarPointsTo(v2, h2).
```

Example: Points-to Analysis

```
// v1 = new …
VarPointsTo(v1, h1) :- New(v1, h1).

// v1 = v2
VarPointsTo(v1, h2) :- Assign(v1, v2),
                       VarPointsTo(v2, h2).

// v1 = v2.f
VarPointsTo(v1, h2) :- Load(v1, v2, f),
                       VarPointsTo(v2, h1),
                       HeapPointsTo(h1, f, h2).

// v1.f = v2
HeapPointsTo(h1, f, h2) :- Store(v1, f, v2),
                           VarPointsTo(v1, h1),
                           VarPointsTo(v2, h2).
```

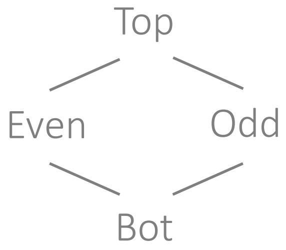


Limitations of Datalog

- No lattices
- No functions
- Poor interoperability

Flix addresses these limitations.

Example: Parity Analysis (1/4)



```
enum Parity {
       case Top,
  case Even, case Odd,
       case Bot
}

fn leq(e1: Parity, e2: Parity): Bool =
  match (e1, e2) with {
    case (Bot, _)        => true
    case (Even, Even)    => true
    case (Odd, Odd)      => true
    case (_, Top)        => true
    case _               => false
  }

fn sum(e1: Parity, e2: Parity): Parity = …

let Parity<> = (Bot, Top, leq, lub, glb);
```

Example: Parity Analysis (2/4)

```
lat A(a: Int, b: Parity<>);

A(1, Even).
A(2, Odd).
A(3, Top).
A(4, x) :- A(1, x).
```

Solution:

Example: Parity Analysis (2/4)

```
lat A(a: Int, b: Parity<>);
```

```
A(1, Even).
A(2, Odd).
A(3, Top).
A(4, x) :- A(1, x).
```

Solution:

```
A(1, Even), A(2, Odd), A(3, Top)
```

Example: Parity Analysis (2/4)

```
lat A(a: Int, b: Parity<>);
```

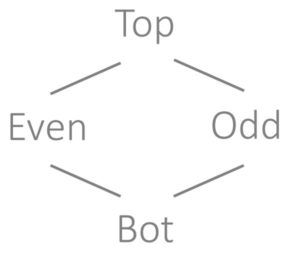
```
A(1, Even).
A(2, Odd).
A(3, Top).
A(4, x) :- A(1, x).
```

Solution:

```
A(1, Even), A(2, Odd), A(3, Top)
A(4, Even)
```

Example: Parity Analysis (3/4)

```
lat B(a: Int, b: Parity<>);

B(1, Even).
B(2, Even).
B(2, Odd).
```



Top

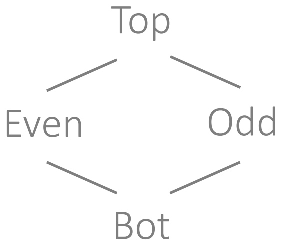
Even Odd

Bot

Solution:

Example: Parity Analysis (3/4)

```
lat B(a: Int, b: Parity<>);

B(1, Even).
B(2, Even).
B(2, Odd).
```



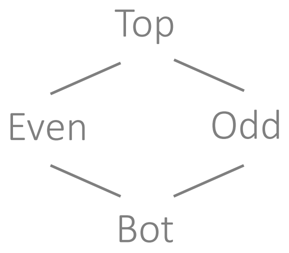
Solution:

```
B(1, Even)
```

Example: Parity Analysis (3/4)

```
lat B(a: Int, b: Parity<>);
```

```
B(1, Even).
B(2, Even).
B(2, Odd).
```



Top

Even

Odd

Bot

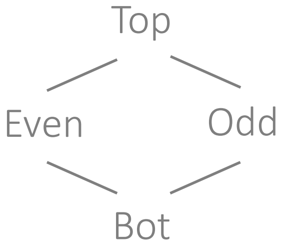
Solution:

```
B(1, Even)
```

```
B(2, Even), B(2, Odd)
```

Example: Parity Analysis (3/4)

```
lat B(a: Int, b: Parity<>);

B(1, Even).
B(2, Even).
B(2, Odd).
```



Top

Even Odd

Bot

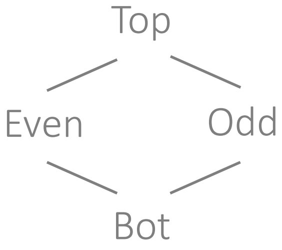
Solution:

```
B(1, Even)
```

~~B(2, Even), B(2, Odd)~~

Example: Parity Analysis (3/4)

```
lat B(a: Int, b: Parity<>);

B(1, Even).
B(2, Even).
B(2, Odd).
```



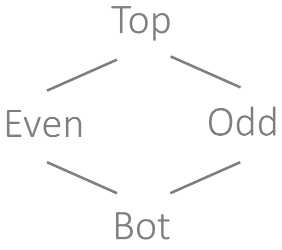
Solution:

B(1, Even)

~~B(2, Even), B(2, Odd)~~ B(2, Even ⊔ Odd)

Example: Parity Analysis (3/4)

```
lat B(a: Int, b: Parity<>);

B(1, Even).
B(2, Even).
B(2, Odd).
```



Top

Even

Odd

Bot

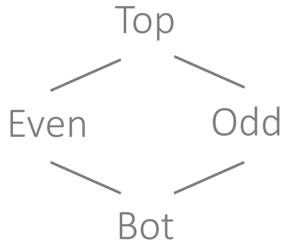
Solution:

B(1, Even)

~~B(2, Even), B(2, Odd)~~ B(2, Top)

Example: Parity Analysis (3/4)

```
lat B(a: Int, b: Parity<>);

B(1, Even).
B(2, Even).
B(2, Odd).
```



Top

Even

Odd

Bot

Solution:

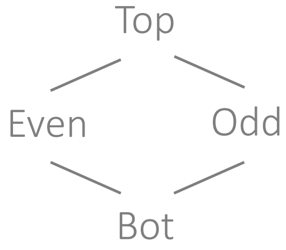
`B(1, Even)`

~~`B(2, Even), B(2, Odd)`~~ `B(2, Top)`

Can we replace `B(1, Even)` with `B(1, Top)`?

Example: Parity Analysis (3/4)

```
lat B(a: Int, b: Parity<>);

B(1, Even).
B(2, Even).
B(2, Odd).
```



Solution:

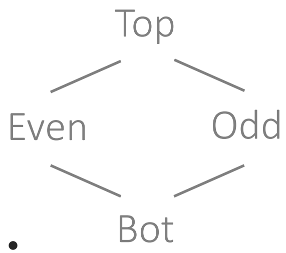
`B(1, Even)`

~~`B(2, Even), B(2, Odd)`~~ `B(2, Top)`

Can we replace `B(1, Even)` with `B(1, Top)`? No.

Example: Parity Analysis (4/4)

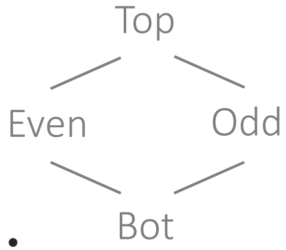
```
lat C(a: Int, b: Parity<>);

C(1, Even).
C(2, Odd).
C(3, sum(x, y)) :- C(1, x), C(2, y).
C(1, Odd).
```



Solution:

Example: Parity Analysis (4/4)

```
lat C(a: Int, b: Parity<>);

C(1, Even).
C(2, Odd).
C(3, sum(x, y)) :- C(1, x), C(2, y).
C(1, Odd).
```



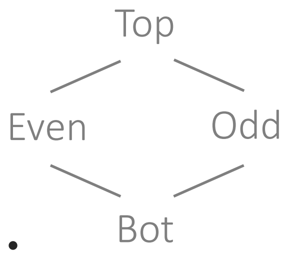
Solution:

```
C(1, Even), C(2, Odd)
```

Example: Parity Analysis (4/4)

```
lat C(a: Int, b: Parity<>);
```

```
C(1, Even).
C(2, Odd).
C(3, sum(x, y)) :- C(1, x), C(2, y).
C(1, Odd).
```



Top

Even

Odd

Bot

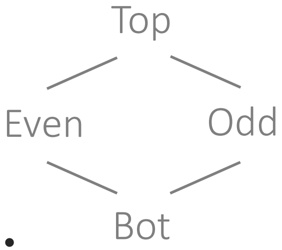
Solution:

```
C(1, Even), C(2, Odd)
```

```
C(3, Odd)
```

Example: Parity Analysis (4/4)

```
lat C(a: Int, b: Parity<>);

C(1, Even).
C(2, Odd).
C(3, sum(x, y)) :- C(1, x), C(2, y).
C(1, Odd).
```



Top

Even

Odd

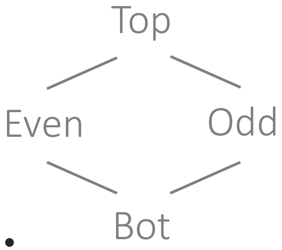
Bot

Solution:

```
C(1, Even), C(2, Odd)

C(3, Odd)

C(1, Odd)
```

Example: Parity Analysis (4/4)

```
lat C(a: Int, b: Parity<>);

C(1, Even).
C(2, Odd).
C(3, sum(x, y)) :- C(1, x), C(2, y).
C(1, Odd).
```



Solution:

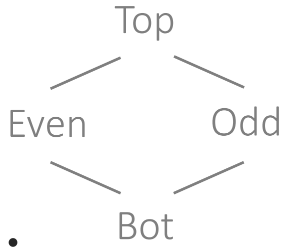
~~C(1, Even)~~, C(2, Odd)

C(3, Odd)

~~C(1, Odd)~~

Example: Parity Analysis (4/4)

```
lat C(a: Int, b: Parity<>);

C(1, Even).
C(2, Odd).
C(3, sum(x, y)) :- C(1, x), C(2, y).
C(1, Odd).
```



Top

Even Odd

Bot

Solution:

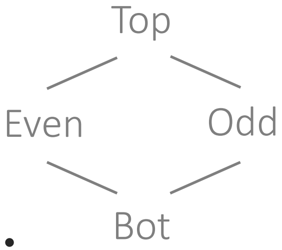
~~C(1, Even)~~, C(2, Odd)

C(3, Odd)

~~C(1, Odd)~~ C(1, Even ⊔ Odd)

Example: Parity Analysis (4/4)

```
lat C(a: Int, b: Parity<>);

C(1, Even).
C(2, Odd).
C(3, sum(x, y)) :- C(1, x), C(2, y).
C(1, Odd).
```



Solution:

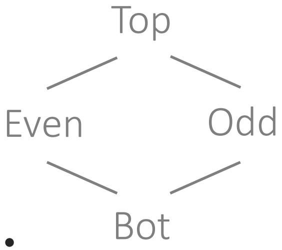
~~C(1, Even)~~, C(2, Odd)

C(3, Odd)

~~C(1, Odd)~~ C(1, Top)

Example: Parity Analysis (4/4)

```
lat C(a: Int, b: Parity<>);

C(1, Even).
C(2, Odd).
C(3, sum(x, y)) :- C(1, x), C(2, y).
C(1, Odd).
```



Solution:

~~C(1, Even)~~, C(2, Odd)

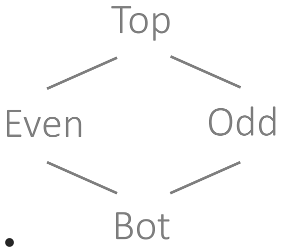
C(3, Odd)

~~C(1, Odd)~~ C(1, Top)

C(3, Top)

Example: Parity Analysis (4/4)

```
lat C(a: Int, b: Parity<>);

C(1, Even).
C(2, Odd).
C(3, sum(x, y)) :- C(1, x), C(2, y).
C(1, Odd).
```



Top

Even

Odd

Bot

Solution:

~~C(1, Even)~~, C(2, Odd)

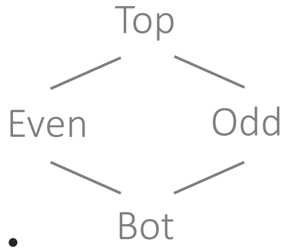
~~C(3, Odd)~~

~~C(1, Odd)~~   C(1, Top)

~~C(3, Top)~~

Example: Parity Analysis (4/4)

```
lat C(a: Int, b: Parity<>);

C(1, Even).
C(2, Odd).
C(3, sum(x, y)) :- C(1, x), C(2, y).
C(1, Odd).
```



Top

Even Odd

Bot

Solution:

~~C(1, Even)~~, C(2, Odd)

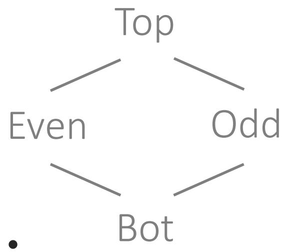
~~C(3, Odd)~~

~~C(1, Odd)~~ C(1, Top)

~~C(3, Top)~~ C(3, Odd ⊔ Top)

Example: Parity Analysis (4/4)

```
lat C(a: Int, b: Parity<>);

C(1, Even).
C(2, Odd).
C(3, sum(x, y)) :- C(1, x), C(2, y).
C(1, Odd).
```



Solution:

~~C(1, Even)~~, C(2, Odd)

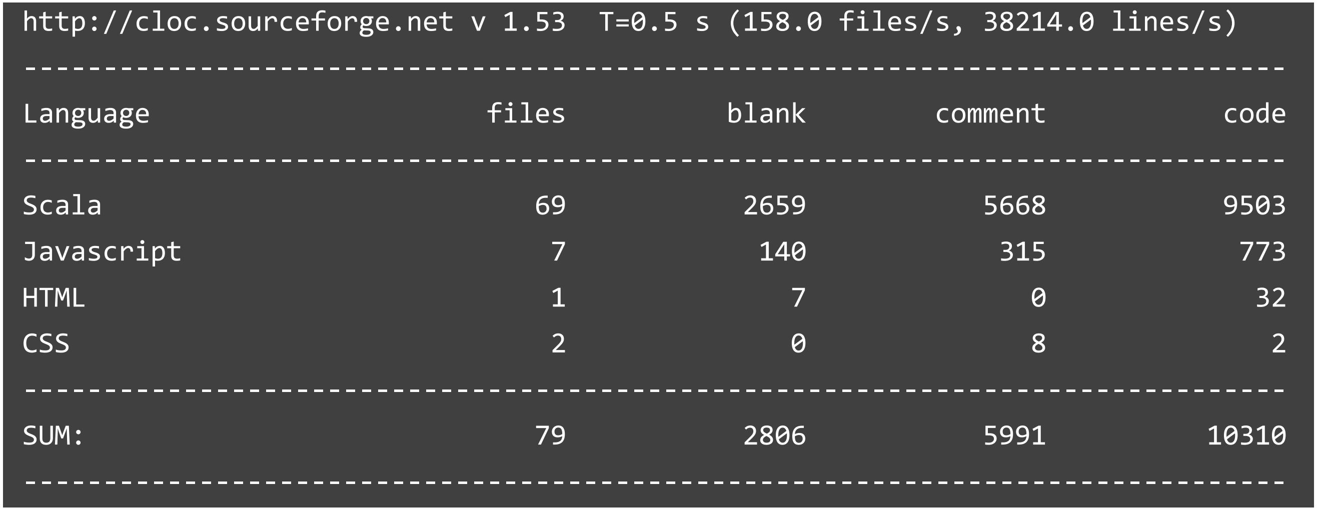
~~C(3, Odd)~~

~~C(1, Odd)~~ C(1, Top)

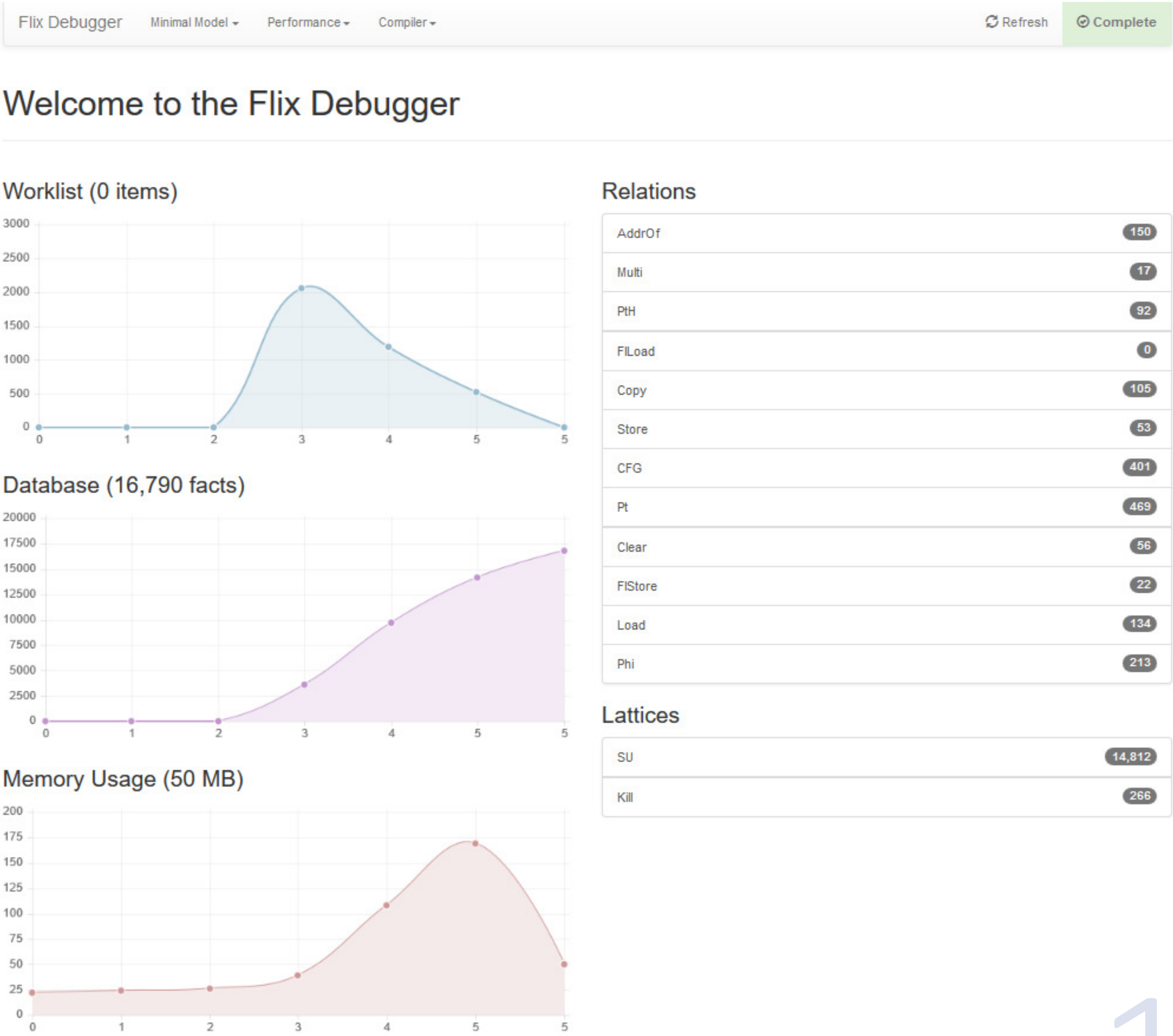
~~C(3, Top)~~ C(3, Top)

Implementation

About 9.5 KLOC of Scala code.



```
http://cloc.sourceforge.net v 1.53  T=0.5 s (158.0 files/s, 38214.0 lines/s)
-------------------------------------------------------------------------------
Language                     files          blank        comment           code
-------------------------------------------------------------------------------
Scala                           69           2659           5668           9503
Javascript                       7            140            315            773
HTML                             1              7              0             32
CSS                              2              0              8              2
-------------------------------------------------------------------------------
SUM:                            79           2806           5991          10310
-------------------------------------------------------------------------------
```



Welcome to the Flix Debugger

Worklist (0 items)

Database (16,790 facts)

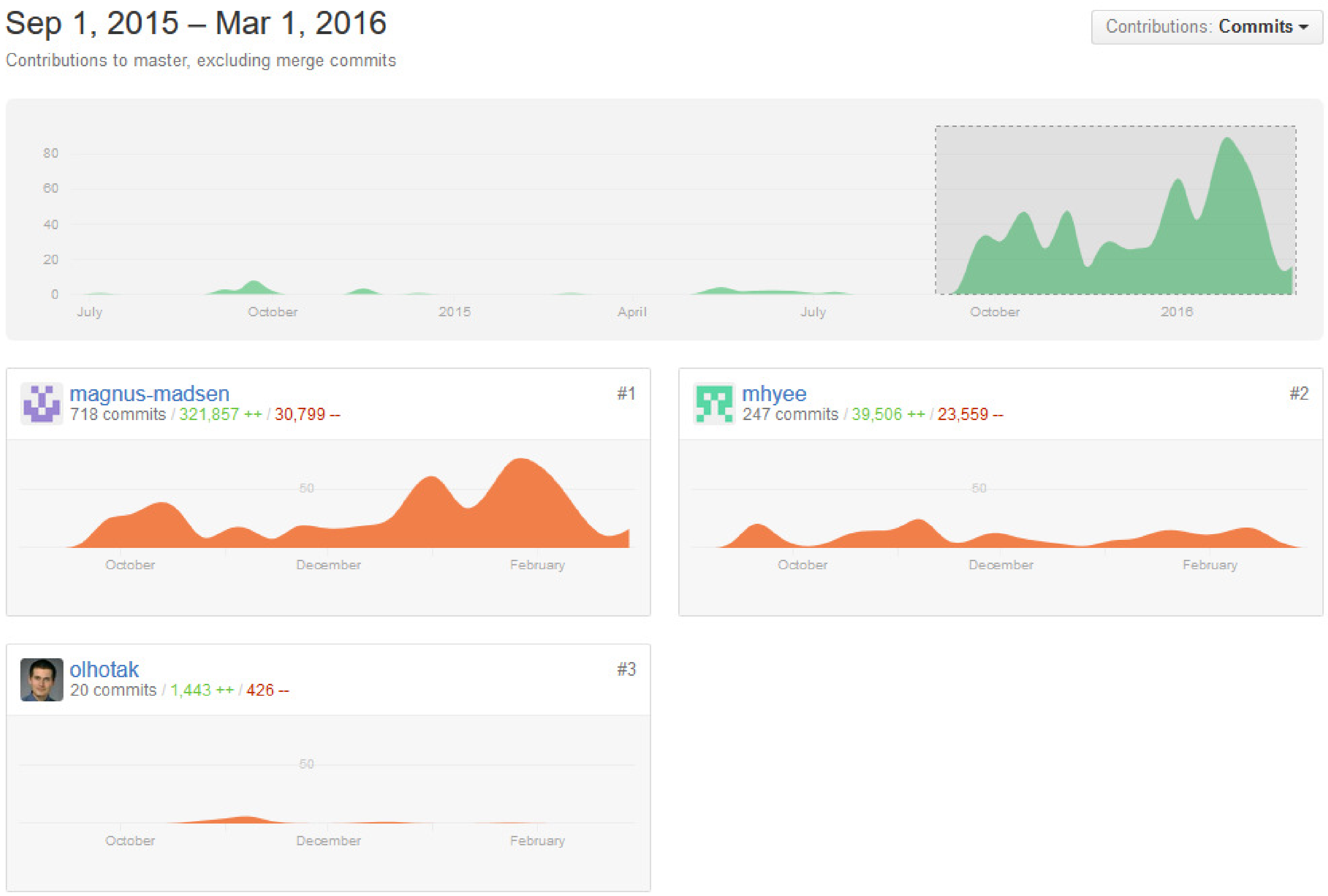
Memory Usage (50 MB)

Relations

AddrOf	150
Multi	17
PtH	92
FILoad	0
Copy	105
Store	53
CFG	401
Pt	469
Clear	56
FIStore	22
Load	134
Phi	213

Lattices

SU	14,812
Kill	266



Sep 1, 2015 – Mar 1, 2016

Contributions to master, excluding merge commits

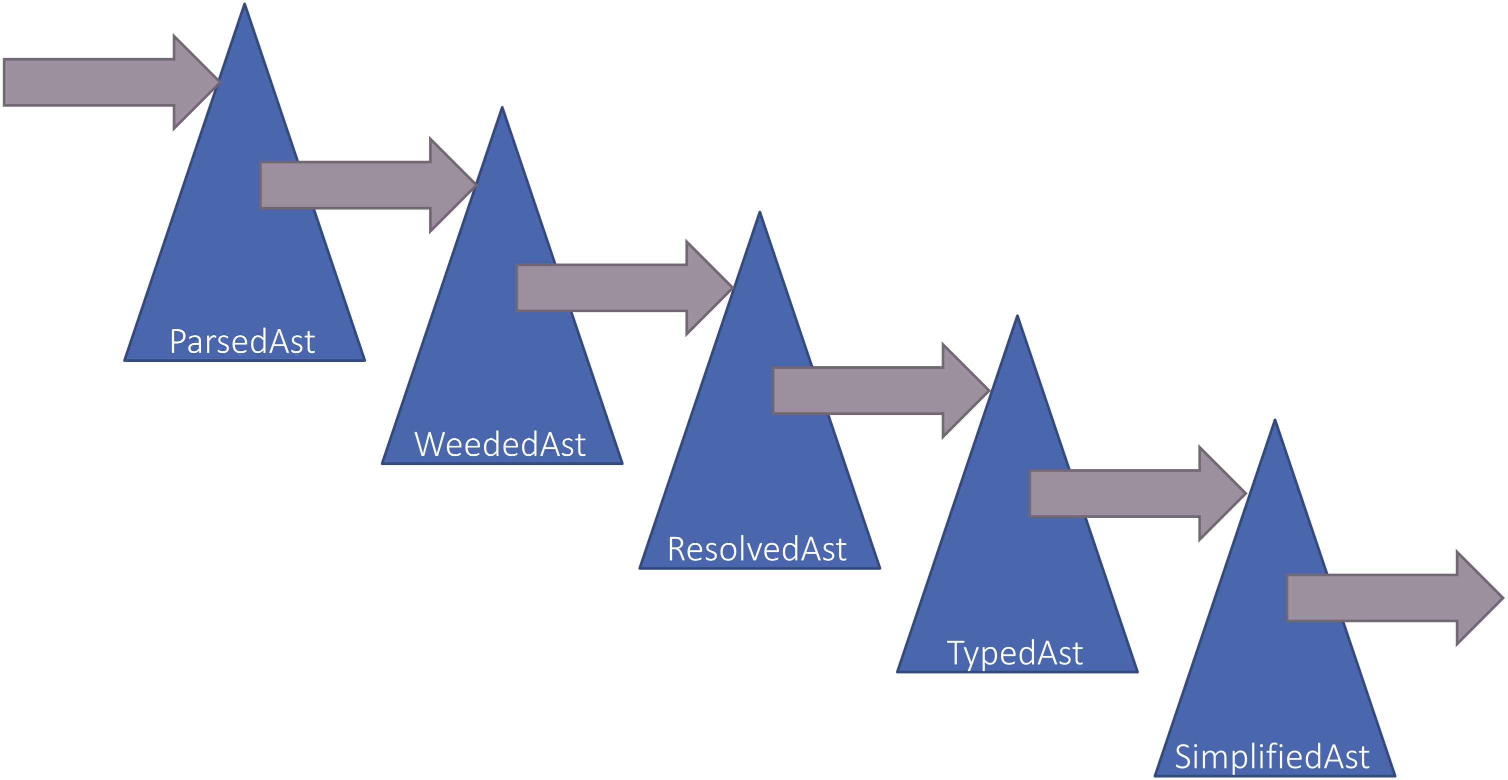
Contributions: **Commits** ▾

magnus-madsen #1
718 commits / 321,857 ++ / 30,799 --

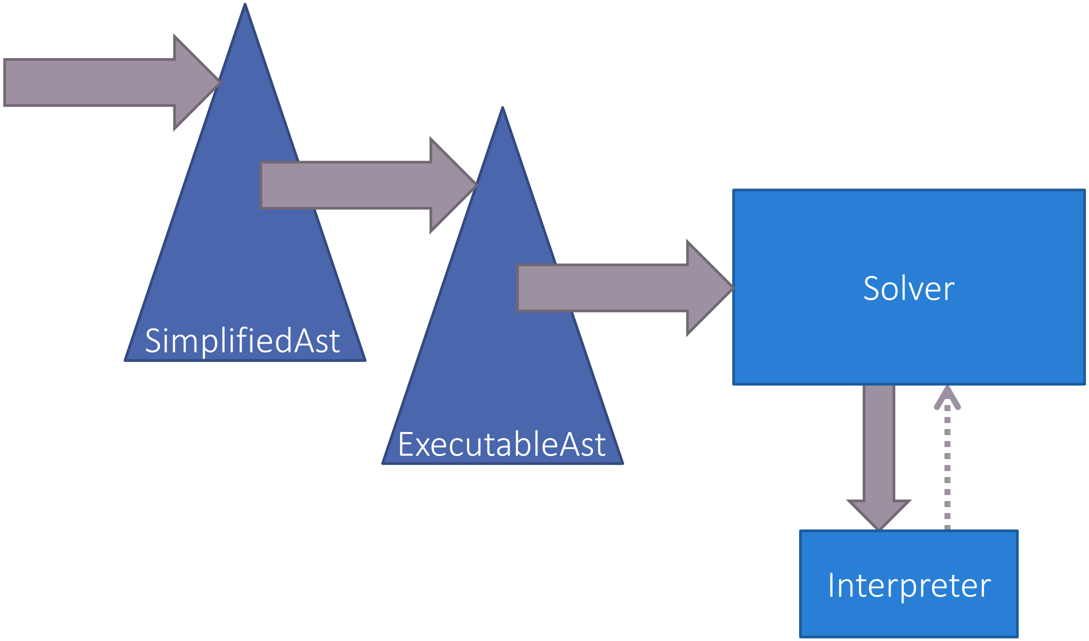
mhyee #2
247 commits / 39,506 ++ / 23,559 --

olhotak #3
20 commits / 1,443 ++ / 426 --

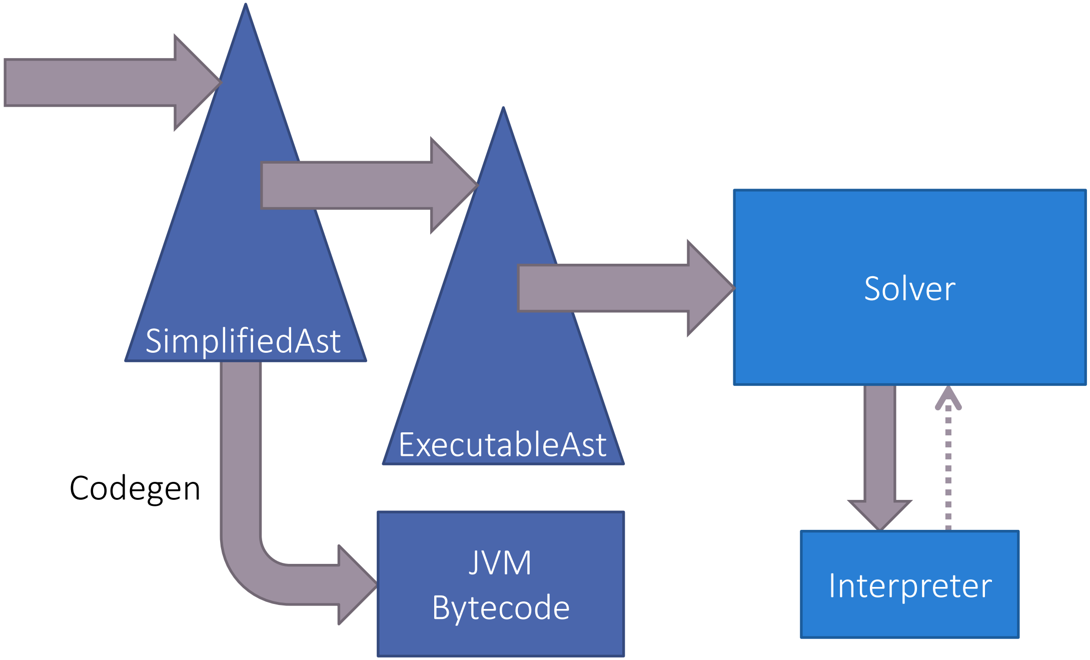
Architecture: Front-End



Architecture: Back-End



Architecture: Back-End



Current and Future Work

Performance

- Code generation
- Optimizations (Luqman Aden)

Safety and Verification

- Integration with Leon (Billy Jin)

Negation

Summary

Flix is a declarative language for solving fixed-point computations on lattices.

Paper: to appear at PLDI 2016.

Future work: performance, safety, and negation.