From Datalog to FLIX: A Declarative Language for Fixed Points on Lattices

Magnus Madsen
University of Waterloo, Canada
mmadsen@uwaterloo.ca

Ming-Ho Yee
University of Waterloo, Canada
ming-ho.yee@uwaterloo.ca

Ondřej Lhoták
University of Waterloo, Canada
olhotak@uwaterloo.ca

http://flix.github.io/
Motivation

Implementation of static analyses is difficult:

• analyses are often interrelated:
  • (e.g. conditional constant propagation)
  • (e.g. points-to analysis and call graph construction)
• requires a complicated arrangement of work lists.

⇒ Hard to ensure correctness.
⇒ Hard to ensure performance/scalability.
⇒ Renewed interest in declarative programming.
What is Declarative Programming?
(aka logic programming)

The **what**, not the **how**.

Find $x$ such that:

\[ 3 + x = 15 \]

⇒ Easy to understand whether we are solving the right problem!
What is Declarative Programming?

Separates the choice of:

• **evaluation strategy** (e.g. work list order)
• **data structures** (e.g. bitsets/hashsets/BDDs)

from the specification of the problem.

⇒ **We can leave these choices up to the solver or override them when needed.**
What is **Datalog**?

A declarative language for constraints on relations:
- Prolog-style rules (horn clauses).
- Successfully used in large-scale points-to analyses
  [Bravenboer et al.], [Smaragdakis et al.]

**Useful theoretical and practical properties:**
- Every Datalog program eventually terminates.
- Every Datalog program has a unique solution.

⇒ **Debugging is easier!**
Examples

• Computing your aunts and uncles:

  \[
  \text{AuntOrUncle}(x, z) :- \text{Parent}(x, y), \text{Brother}(y, z).
  \]
  \[
  \text{AuntOrUncle}(x, z) :- \text{Parent}(x, y), \text{Sister}(y, z).
  \]

• Computing the transitive closure of a graph:

  \[
  \text{Path}(x, z) :- \text{Path}(x, y), \text{Edge}(y, z).
  \]
What we **can** and **can't** do in Datalog:

**Analyses based on relations:**
- Points-To
- Definite Assignment
- Reaching Definitions
- ...

**Analyses based on lattices:**
- Sign Analysis
- Constant Propagation
- Interval Analysis
- ...
Example: Constant Propagation

What Datalog has:

What we wanted:

⇒ We need lattices.
⇒ We need functions.

fixed finite set

infinite set
Introducing Flix

A blend of Logic and Functional programming.

- Inspired by Datalog.
- User-defined lattices.
- User-defined monotone filter and transfer functions.

- Interoperates with languages on the JVM.
The Anatomy of a **Datalog** Rule

\[ H(\vec{t}) \leftarrow B(\vec{t}), \ldots, B(\vec{t}). \]

**Head**

**Predicates**

**Body**

Terms: Variables or Constants
The Anatomy of a Flix Rule

Filter Function

\[ H_\ell (\bar{t}, f(\bar{t})) \leftarrow \varphi(\bar{t}), B_\ell (\bar{t}), \ldots, B_\ell (\bar{t}). \]

Transfer Function
Datalog vs. Flix

The **Datalog** program:

\[
\begin{align*}
A(\text{"foo"}). \\
A(\text{"bar"}).
\end{align*}
\]

has the minimal model:

\[
\{A(\text{"foo"}), A(\text{"bar"})\}
\]

The **Flix** program:

\[
\begin{align*}
A(\text{Cst}(1)). \\
A(\text{Cst}(2)). \\
B(\text{Cst}(3)).
\end{align*}
\]

has the minimal model:

\[
\{A(\text{Top}), B(\text{Cst}(3))\}\]
Flix Semantics

The Flix program:

\[ A(\text{Cst}(1)). \quad B(\text{Cst}(2)). \]
\[ R(x) : - \quad A(x). \]
\[ R(x) : - \quad B(x). \]

has the minimal model:

\{ A(\text{Cst}(1)), \]
\{ B(\text{Cst}(2)), \]
\{ R(\text{Top}) \} \]

The Flix program:

\[ A(\text{Cst}(1)). \quad B(\text{Cst}(2)). \]
\[ R(x) : - \quad A(x), \quad B(x). \]

has the minimal model:

\{ A(\text{Cst}(1)), \]
\{ B(\text{Cst}(2)) \}
Constant Propagation

Input Relations:

\[ \text{rel AddExp}(r: \text{Var}, x: \text{Var}, y: \text{Var}) \quad r = x + y \]
\[ \text{rel DivExp}(r: \text{Var}, x: \text{Var}, y: \text{Var}) \quad r = x / y \]

Computed Lattices:

\[ \text{lat LocalVar}(x: \text{Var}, v: \text{Constant}) \]

- key
- lattice
Lattice Definition

```python
enum Constant {
    case Top,
    case Cst(Int),
    case Bot
}
```
def leq(e1: Constant, e2: Constant): Bool
    = match (e1, e2) with {
        case (Bot, _) => true
        case (Cst(n1), Cst(n2)) => n1 == n2
        case (_, Top) => true
        case _ => false
    }

And define lub and glb in similar way...
Analysis Rules

\texttt{LocalVar}(r, \texttt{sum}(x, y)) :- \texttt{AddExp}(r, v1, v2),
\texttt{LocalVar}(v1, x),
\texttt{LocalVar}(v2, y).

\texttt{LocalVar}(r, \texttt{div}(x, y)) :- \texttt{DivExp}(r, v1, v2),
\texttt{LocalVar}(v1, x),
\texttt{LocalVar}(v2, y).
def \texttt{sum}(e1: \texttt{Constant}, e2: \texttt{Constant}): \texttt{Constant} \\
  = \texttt{match} \ (e1, e2) \ \texttt{with} \ { \\
    \text{case} \ (_\text{,} \ \text{Bot}) \ \Rightarrow \ \text{Bot} \\
    \text{case} \ (\text{Bot}\text{,} \ _) \ \Rightarrow \ \text{Bot} \\
    \text{case} \ (\text{Cst}(n1), \ \text{Cst}(n2)) \ \Rightarrow \ \text{Cst}(n1 + n2) \\
    \text{case} \ _ \ \Rightarrow \ \text{Top} \\
  }
Example: Finding Bugs

ArithmeticError(r) :- isMaybeZero(y),
DivExp(r, n, d),
LocalVar(d, y).
def isMaybeZero(e: Constant): Bool
    = match e with { 
        case Bot => false
        case Cst(n) => n == 0
        case Top => true
    }
Experiments

We have expressed three analyses in Flix:

• The Strong Update Analysis [Lhoták and Chung]
• The IFDS algorithm [Reps, Horwitz, and Sagiv]
  • Instantiated with the Alias-Set analysis [Naeem and Lhoták]
• The IDE algorithm [Sagiv, Reps, and Horwitz]
Strong Update Analysis

• Hybrid points-to analysis for C programs:
  • flow-sensitive for singleton points-to sets.
  • flow-insensitive for everything else.
Strong Update Analysis

\[ \ell : p = &a \quad \{a\} \subseteq pt(p) \quad \text{[ADDROF]} \]
\[ \ell : p = q \quad pt(q) \subseteq pt(p) \quad \text{[COPY]} \]
\[ \ell : *p = q \quad \forall a \in pt(p) . pt(q) \subseteq su[\ell](a) \quad \text{[STORE]} \]
\[ \ell : p = *q \quad \forall a \in pt(q) . ptsu[\ell](a) \subseteq pt(p) \quad \text{[LOAD]} \]
\[ \ell_1 \in \text{pred}(\ell_2) \quad \forall a \in A . su[\ell_1](a) \subseteq su[\ell_2](a) \quad \text{[CFLOW]} \]
\[ \ell \in \mathcal{L} \quad \forall a \in A \setminus \text{kill}(\ell) . su[\ell](a) \subseteq su[\ell](a) \quad \text{[PRESERVE]} \]

Where \( ptsu[\ell](a) \triangleq \begin{cases} su[\ell](a) & \text{if } su[\ell](a) \neq T \\ pt(a) & \text{if } su[\ell](a) = T \end{cases} \)
Strong Update Analysis

\[
\begin{align*}
\text{Pt}(p, a) & : \text{AddrOf}(p, a). & \text{[AddrOf]} \\
\text{Pt}(p, a) & : \text{Copy}(p, q), \text{Pt}(q, a). & \text{[Copy]} \\
\text{SUAfter}(l, a, \text{Single}(b)) & : \\
& \text{Store}(l, p, q), \text{Pt}(p, a), \text{Pt}(q, b). & \text{[Store]} \\
\text{PtH}(a, b) & : \text{Store}(l, p, q), \text{Pt}(p, a), \text{Pt}(q, b). & \text{[Load]} \\
\text{Pt}(p, b) & : \text{Load}(l, p, q), \text{Pt}(q, a), \text{PtSU}(l, a, b). & \text{[CFlow]} \\
\text{SUBBefore}(l_2, a, t) & : \text{CFG}(l_1, l_2), \text{SUAfter}(l_1, a, t). & \text{[Preserve]} \\
\text{SUAfter}(l, a, t) & : \text{SUBBefore}(l, a, t), \text{Preserve}(l, a). & \text{[Preserve]} \\
\text{PtSU}(l, a, b) & : \text{PtH}(a, b), \text{SUBBefore}(l, a, t), \text{filter}(t, b). & \text{} \\
\end{align*}
\]
IFDS & IDE

Inter-procedural context-sensitive dataflow analyses:

• expressed as graph reachability problems.
• Interprocedural Finite Distributive Subset (IFDS)
  • pure graph reachability.
• Inteprocedural Distributive Environments (IDE)
  • IFDS with composition of micro-functions along the path.

• Anecdotally, these algorithms are hard to understand.
IFDS

• **Input:** the exploded super-graph.
  • The super-graph is the inter-procedural CFG.
  • The exploded super-graph is a copy of the CFG for each analysis element (in the distributive subset).

• **Output:** *Path Edges* + *Summary Edges*. 
IFDS – Graphical Formulation

- Exploded Super-Graph
- Path Edge
- Summary Edge
- Inferred Edge

Diagram showing the graphical formulation of IFDS with nodes labeled n, m, s, and e, and arrows indicating different types of edges.
IFDS

PathEdge(d1, m, d3) :-
  PathEdge(d1, n, d2),
  CFG(n, m),
  d3 <- eshIntra(n, d2).

- The CFG is represented as a *tabulated relation*.
- The exploded super-graph (eshIntra) must be represented as a *function computed on-demand*. 

(node, element)-pair
IFDS

PathEdge(d1, m, d3) :-
  CFG(n, m),
  PathEdge(d1, n, d2),
  d3 <- eshIntra(n, d2).
PathEdge(d1, m, d3) :-
  CFG(n, m),
  PathEdge(d1, n, d2),
  SummaryEdge(n, d2, d3).
PathEdge(d3, start, d3) :-
  PathEdge(d1, call, d2),
  CallGraph(call, target),
  EshCallStart(call, d2, target, d3),
  StartNode(target, start).
SummaryEdge(call, d4, d5) :-
  CallGraph(call, target),
  StartNode(target, start),
  EndNode(target, end),
  EshCallStart(call, d4, target, d1),
  PathEdge(d1, end, d2),
  d5 <- eshEndReturn(target, d2, call).
EshCallStart(call, d, target, d2) :-
  PathEdge(_, call, d),
  CallGraph(call, target),
  d2 <- eshCallStart(call, d, target).
Result(n, d2) :-
  PathEdge(_, n, d2).

IDE

JumpFn(d1, m, d3, comp(long, short)) :-
  CFG(n, m),
  JumpFn(d1, n, d2, long),
  (d3, short) <- eshIntra(n, d2).
JumpFn(d1, m, d3, comp(caller, summary)) :-
  CFG(n, m),
  JumpFn(d1, n, d2, caller),
  SummaryFn(n, d2, d3, summary).
JumpFn(d3, start, d3, identity()) :-
  JumpFn(d1, call, d2, _),
  CallGraph(call, target),
  EshCallStart(call, d2, target, d3, _),
  StartNode(target, start).
SummaryFn(call, d4, d5, comp(comp(cs, se), er)) :-
  CallGraph(call, target),
  StartNode(target, start),
  EndNode(target, end),
  EshCallStart(call, d4, target, d1, cs),
  JumpFn(d1, end, d2, se),
  (d5, er) <- eshEndReturn(target, d2, call).
EshCallStart(call, d, target, d2, cs) :-
  JumpFn(_, call, d, _),
  CallGraph(call, target),
  (d2, cs) <- eshCallStart(call, d, target).
InProc(p, start) :- StartNode(p, start).
InProc(p, m) :- InProc(p, n), CFG(n, m).
Result(n, d, apply(fn, vp)) :-
  ResultProc(proc, dp, vp),
  InProc(proc, n),
  JumpFn(dp, n, d, fn).
ResultProc(proc, dp, apply(cs, v)) :-
  Result(call, d, v),
  EshCallStart(call, d, proc, dp, cs).
The Flix Programming Language http://flix.github.io — Edit

- 3,192 commits
- 2 branches
- 2 releases
- 4 contributors

Branches: master

<table>
<thead>
<tr>
<th>Branch</th>
<th>Description</th>
<th>Latest commit</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>master</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Files:

- **doc/cheatsheet**: Added contributors. Delete CLAs.
- **examples**: Fix parser error in TestVerifier test (in no longer a keyword)
- **lib**: Upgrade ScalaTest to 2.2.6
- **library**: WIP library
- **main**: Print out the N and seed value. Also use Map instead of HashMap a...
- **.gitignore**: add target to .gitignore
- **CONTRIBUTORS.md**: add Ondřej Lholák to contributors
- **LICENSE.md**: Added license.
- **README.md**: WIP namespaces.
- **build.sbt**: Upgrade ScalaTest to 2.2.6

The Flix Programming Language

Main repository for the source code of the Flix compiler and run-time.

See the official Flix website for more information.

Reporting Bugs & Feature Requests

You are most welcome to report bugs or request features on this GitHub page.
Flix. Functional. Logical.

The elegance of functional programming with the conciseness of logic programming.
Think SQL, but on steroids.
Welcome to the Flix Debugger

Worklist (2,130 items)

Database (402,530 facts)

Memory Usage (242 MB)

Relations

<table>
<thead>
<tr>
<th>Relation</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>/PIH</td>
<td>292</td>
</tr>
<tr>
<td>/Phi</td>
<td>2,702</td>
</tr>
<tr>
<td>/Store</td>
<td>316</td>
</tr>
<tr>
<td>/Copy</td>
<td>481</td>
</tr>
<tr>
<td>/Clear</td>
<td>223</td>
</tr>
<tr>
<td>/CFG</td>
<td>4,523</td>
</tr>
<tr>
<td>/FLoad</td>
<td>8</td>
</tr>
<tr>
<td>/FISStore</td>
<td>60</td>
</tr>
<tr>
<td>/Pl</td>
<td>4,124</td>
</tr>
<tr>
<td>/AddrOf</td>
<td>915</td>
</tr>
<tr>
<td>/Load</td>
<td>2,130</td>
</tr>
<tr>
<td>/Multi</td>
<td>148</td>
</tr>
</tbody>
</table>

Lattices

<table>
<thead>
<tr>
<th>Relation</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>/SU</td>
<td>390,378</td>
</tr>
<tr>
<td>/Kill</td>
<td>2,967</td>
</tr>
</tbody>
</table>
Performance / Rules

The table shows the number of milliseconds spent in evaluation of each rule.

<table>
<thead>
<tr>
<th>Location</th>
<th>Rule</th>
<th>Hits</th>
<th>Total Time (msec)</th>
<th>Latency (msec/op)</th>
<th>Throughput (ops/msec)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Suopt.flix:102:1</td>
<td>SU(l1,a,t) :- CFG(l1, l1), SU(l1,a,t), killNot(a, k), Kill(l1,k).</td>
<td>910,300</td>
<td>18,582 msec</td>
<td>0.0202 msec/op</td>
<td>49 ops/msec</td>
</tr>
<tr>
<td>Suopt.flix:86:1</td>
<td>Pt(p,b) :- Load(l,p,q), Pt(q,a), filter(t, b), Pth(a,b), SU(l,a,t).</td>
<td>930,051</td>
<td>3,745 msec</td>
<td>0.0040 msec/op</td>
<td>248 ops/msec</td>
</tr>
<tr>
<td>Suopt.flix:101:1</td>
<td>SU(l2,a,t) :- CFG(l1, l2), SU(l1,a,t), Multi(a).</td>
<td>915,193</td>
<td>3,245 msec</td>
<td>0.0035 msec/op</td>
<td>282 ops/msec</td>
</tr>
<tr>
<td>Suopt.flix:112:1</td>
<td>SU(l1,a,f(b)) :- Clear(l), Pth(a,b).</td>
<td>810</td>
<td>784 msec</td>
<td>0.9679 msec/op</td>
<td>1 ops/msec</td>
</tr>
<tr>
<td>Suopt.flix:118:1</td>
<td>Kill(l, top(42)) :- Phi(1,_,__).</td>
<td>1</td>
<td>133 msec</td>
<td>133.0000 msec/op</td>
<td>0 ops/msec</td>
</tr>
<tr>
<td>Suopt.flix:79:1</td>
<td>SU(l1,a,f(b)) :- Store(l,p,q), Pt(p,a), Pt(q,b).</td>
<td>28,099</td>
<td>62 msec</td>
<td>0.0022 msec/op</td>
<td>453 ops/msec</td>
</tr>
<tr>
<td>Suopt.flix:81:1</td>
<td>Pth(a,b) :- Store(l,p,q), Pt(p,a), Pt(q,b).</td>
<td>28,099</td>
<td>50 msec</td>
<td>0.0018 msec/op</td>
<td>562 ops/msec</td>
</tr>
<tr>
<td>Suopt.flix:87:1</td>
<td>Pt(p,b) :- FLoad(p,q,_), Pt(q,a), Pth(a,b).</td>
<td>14,859</td>
<td>27 msec</td>
<td>0.0018 msec/op</td>
<td>550 ops/msec</td>
</tr>
<tr>
<td>Suopt.flix:82:1</td>
<td>Pth(a,b) :- FStore(p,q,_), Pt(p,a), Pt(q,b).</td>
<td>28,099</td>
<td>25 msec</td>
<td>0.0009 msec/op</td>
<td>1,124 ops/msec</td>
</tr>
<tr>
<td>Suopt.flix:74:1</td>
<td>Pt(p,a) :- Copy(p,q), Pt(q,a).</td>
<td>14,850</td>
<td>23 msec</td>
<td>0.0010 msec/op</td>
<td>611 ops/msec</td>
</tr>
<tr>
<td>Suopt.flix:70:1</td>
<td>Pt(p,a) :- AddrOf(p,a).</td>
<td>1</td>
<td>17 msec</td>
<td>17.0000 msec/op</td>
<td>0 ops/msec</td>
</tr>
<tr>
<td>Suopt.flix:117:1</td>
<td>Kill(l,f(b)) :- Store(l,p,q), Pt(p,b).</td>
<td>14,050</td>
<td>14 msec</td>
<td>0.0010 msec/op</td>
<td>1,004 ops/msec</td>
</tr>
</tbody>
</table>
What's in the paper: \textbf{MATH} (just a bit ...)

- From Datalog to Flix semantics:
  - explains the relationship between Datalog and Flix.
  - develops the model-theoretic semantics of Flix.

- Presents semi-naïve evaluation for Flix:
  - the basis for efficient Datalog and Flix solvers.

- Experimental results and details of analyses:
  - the Strong Update analysis, and
  - the IFDS Alias-Set analysis.
Summary: Flix!

• A new declarative and functional programming language for fixed point computations on lattices.
• Inspired by Datalog and extended with lattices and monotone transfer/filter functions.
• Implementation freely available:  
  http://github.com/flix
• Documentation and more information:  
  http://flix.github.io

Thank You!
This page intentionally left blank