Demanded

Abstract Interpretation

Benno Stein
University of Colorado Boulder
benno.stein@colorado.edu

Bor-Yuh Evan Chang
University of Colorado Boulder & Amazon
evan.chang@colorado.edu

Manu Sridharan
University of California, Riverside
manu@cs.ucr.edu

* Bor-Yuh Evan Chang holds concurrent appointments as an Associate Professor at the University of Colorado Boulder and as an Amazon Scholar. This talk describes work performed at the University of Colorado Boulder and is not associated with Amazon.
Batch Static Analysis Workflow
Batch Static Analysis Workflow

Is this program okay?*

* i.e. safe with respect to some program analyzer
Batch Static Analysis Workflow

* i.e. safe with respect to some program analyzer

Is this program okay?*
Batch Static Analysis Workflow

* i.e. safe with respect to some program analyzer
Batch Static Analysis Workflow

Programmer

Is this program okay?

CI Server
Batch Static Analysis Workflow

Is this program okay?

Programmer

CI Server
Batch Static Analysis Workflow

Programmer

Is this program okay?

How about this one?

CI Server
Batch Static Analysis Workflow

Programmer

Is this program okay?

CI Server

How about this one?
Batch Static Analysis Workflow

Programmer

Is this program okay?

How about this one?

CI Server
Batch Static Analysis Workflow

Programmer

Is this program okay?

CI Server

How about this one?

•

•

•

•
Batch Static Analysis Workflow

Programmer

CI Server

Is this program okay?

How about this one?

Commit, merge, deploy, etc.

...
Batch Static Analysis Workflow

Is this program okay?

How about this one?

Programmer

CI Server

Commit, merge, deploy, etc.

This is slow and frustrating: each iteration can be on the order of an hour or more.
Interactive Static Analysis Workflow

Is this program okay?
Interactive Static Analysis Workflow

Is this program okay?
Interactive Static Analysis Workflow

Is this program okay?

Alarms:
- Possible issue Foo at location X
- Possible issue Bar at location Y
- Possible issue Baz at location Z
Interactive Static Analysis Workflow

Did my edits fix those alarm(s)?

Is this program okay?

Alarms:
- Possible issue Foo at location X
- Possible issue Bar at location Y
- Possible issue Baz at location Z
Interactive Static Analysis Workflow

Programmer

Did my edits fix those alarm(s)?

Is this program okay?

CI Server

Alarms:
- Possible issue Foo at location X
- Possible issue Bar at location Y
- Possible issue Baz at location Z
Interactive Static Analysis Workflow

Did my edits fix those alarm(s)?

Is this program okay?

Alarms:
- Possible issue *Foo* at location X
- Possible issue *Bar* at location Y
- Possible issue *Baz* at location Z

Programmer

CI Server
Interactive Static Analysis Workflow

- Did my edits fix those alarm(s)?
- Is this program okay?
- Alarms:
  - Possible issue Foo at location X
  - Possible issue Bar at location Y
  - Possible issue Baz at location Z

Commit, merge, deploy, etc.
Interactive Static Analysis Workflow

Did my edits fix those alarm(s)?

Is this program okay?

User wants this to be:
- **Correct** with respect to underlying batch program analyzer
- **Fast** enough for interactive use as the programmer makes *edits* and issues *queries* — on the order of seconds

Alarms:
- Possible issue *Foo* at location *X*
- Possible issue *Bar* at location *Y*
- Possible issue *Baz* at location *Z*

Commit, merge, deploy, etc.
Interactive Static Analysis Workflow

Programmer

Did my edits fix those alarm(s)?

Commit, merge, deploy, etc.

CI Server

Is this program okay?

Alarms:
- Possible issue Foo at location X

User wants this to be:
- Correct with respect to underlying batch program analyzer
- Fast enough for interactive use as the programmer makes edits and issues queries — on the order of seconds

Analysis designer wants this to be:
- Plug-and-play without need for ad-hoc incremental/demand reasoning
- General w.r.t. domains, handling infinite-height, widening, etc.
Batch Analysis

Program → Over-Approximate Facts
Incremental Analysis
Incremental Analysis

Program

\[\Delta\]

New Program

Over-Approximate Facts
Incremental Analysis

Program

New Program

Over-Ap proximate Facts

New Over-Ap proximate Facts
Incremental Analysis

Program

New Program

Over-Approximate Facts

New Over-Approximate Facts

???
Incremental Analysis
Demand-Driven Analysis

Program

Over-Approximate Facts
Demand-Driven Analysis

Program

query

Over-Accurate Facts
Demand-Driven Analysis

Program query

Over-Approximate Facts result
Demand-Driven Analysis

Program

query

Over-
Approximate
Facts

result

???
Abstract Interpretation

Program $\Sigma^\# \downarrow \varphi_0 \subseteq [[\cdot]]^\# \sqcup$

Over-Approximate Facts
Abstract Interpretation

Program → Abstract Domain → Over-Approximate Facts
Abstract Interpretation

Program → Abstract Domain → Fixed-Point Solver → Over-Approximate Facts
Abstract Interpretation

Pros:
- Expressivity of abstract domains
- Robust metatheory: soundness, termination, etc.
- Modularity w.r.t abstractions and solvers

Program → Fixed-Point Solver → Abstract Domain → Over-Approximate Facts
Abstract Interpretation

Pros:
• Expressivity of abstract domains
• Robust metatheory: soundness, termination, etc.
• Modularity w.r.t abstractions and solvers

Cons:
• Whole-program fixed-points are expensive to compute
• Black-box: invariant map only holds meaning at fixed-point

Program → Fixed-Point Solver → Over-Approximate Facts
Demanded Abstract Interpretation
Demanded Abstract Interpretation

Demand-driven queries
“What is the abstract state at this program location?”
Demanded Abstract Interpretation

Demand-driven queries
“What is the abstract state at this program location?”

Incremental edits
Insert, modify, or delete statements
Demanded Abstract Interpretation

**Demand-driven** queries
“What is the abstract state at this program location?”

**Incremental** edits
Insert, modify, or delete statements

**From-scratch consistent** results
Query responses are guaranteed identical to batch analysis on current program version
Demanded Abstract Interpretation Graphs (DAIGs)

Abstract Domain

Demand-driven queries
“What is the abstract state at this program location?”

Incremental edits
Insert, modify, or delete statements

From-scratch consistent results
Query responses are guaranteed identical to batch analysis on current program version

Initial Program

DAIG
Demanded Abstract Interpretation Graphs (DAIGs)

A DAIG reifies the *dependency* structure of an abstract interpretation.

*Demand-driven* queries
“What is the abstract state at this program location?”

*Incremental* edits
Insert, modify, or delete statements

*From-scratch consistent* results
Query responses are guaranteed identical to batch analysis on current program version
A DAIG reifies the *dependency* structure of an abstract interpretation.

Reference cell vertices...
... contain intermediate analysis results & program syntax
... are uniquely named and potentially empty

**Demand-driven** queries
“What is the abstract state at this program location?”

**Incremental** edits
Insert, modify, or delete statements

**From-scratch consistent** results
Query responses are guaranteed identical to batch analysis on current program version
A DAIG reifies the *dependency* structure of an abstract interpretation.

Reference cell vertices...
... contain intermediate analysis results & program syntax
... are uniquely named and potentially empty

Computation edges...
... *acyclically* connect reference cells
... denote analysis computations (e.g. $[\cdot]$, $\sqcup$, $\triangledown$)

**Demand-driven** queries
“What is the abstract state at this program location?”

**Incremental** edits
Insert, modify, or delete statements

**From-scratch consistent** results
Query responses are guaranteed identical to batch analysis on current program version
Contributions

A DAIG reifies the *dependency* structure of an abstract interpretation.

Reference cell vertices...
... contain program syntax & intermediate analysis results
... are uniquely *named* and potentially empty

Computation edges...
... acyclically connect reference cells
... are labelled by analysis functions (e.g.

\[
\text{\#}, \sqcup, \nabla
\]

(Initial) Program

**Finite & acyclic** dependency graph, supporting incremental and demand-driven evaluation

*Incremental* edits
Insert, modify, or delete statements

*From-scratch consistent* results
Query responses are guaranteed identical to batch analysis on current program version
Contributions

A DAIG reifies the dependency structure of an abstract interpretation. Reference cell vertices... contain program syntax & intermediate analysis results... are uniquely named and potentially empty. Computation edges... acyclically connect reference cells... are labelled by analysis functions (e.g. $\#$, $\sqcup$, $\nabla$).

**Cyclic and unbounded** for fixed-point computations in infinite-height domains. Finite & acyclic dependency graph, supporting incremental and demand-driven evaluation.

(Initial) Program

Incremental edits
Insert, modify, or delete statements

From-scratch consistent results
Query responses are guaranteed identical to batch analysis on current program version.
Contributions

A DAIG reifies the *dependency* structure of an abstract interpretation. Reference cell vertices...
... contain program syntax & intermediate analysis results... are uniquely named and potentially empty.
Computation edges...
... acyclically connect reference cells... are labelled by analysis functions (e.g. \(\#\), \(\sqcup\), \(\nabla\)).

**Cyclic and unbounded** for fixed-point computations in infinite-height domains.

**Finite & acyclic** dependency graph, supporting incremental and demand-driven evaluation via *demanded unrolling* of fixed-point computations.

(Initial) Program

**Incremental** edits
Insert, modify, or delete statements

**From-scratch consistent** results
Query responses are guaranteed identical to batch analysis on current program version
A DAIG reifies the dependency structure of an abstract interpretation.

**Contributions**

- **Cyclic** and **unbounded** for fixed-point computations in infinite-height domains.
- **Finite** & **acyclic** dependency graph, supporting incremental and demand-driven evaluation

via **demanded unrolling** of fixed-point computations

- **From-scratch consistent** results
  - Query responses are guaranteed identical to batch analysis on current program version
Demanded Abstract Interpretation Graphs (DAIGs)
Demanded Abstract Interpretation Graphs (DAIGs)
CFG $\rightarrow$ DAIG Encoding (by example)
CFG → DAIG Encoding (by example)
CFG → DAIG Encoding (by example)

Program snippet

Corresponding DAIG snippet

Corresponding DAIG snippet
CFG $\rightarrow$ DAIG Encoding (by example)

Underlined terms are names derived e.g. by hashing
CFG → DAIG Encoding (by example)
CFG $\rightarrow$ DAIG Encoding (by example)
Demand-Driven Query Evaluation

Program snippet

\[ l \rightarrow l' \]

\[ s \rightarrow \epsilon \]

\[ l' \rightarrow s \]
Demand-driven query:
“What is the abstract state at \( l' \)?”
i.e. “What is the value of cell \( l' \)?”
Demand-driven query:
“What is the abstract state at $l'$?”
i.e. “What is the value of cell $l'$?”

$l' = \llbracket l \rightarrow l' \rrbracket^# l$
Demand-driven query:
“What is the abstract state at $l'$?”
i.e. “What is the value of cell $l'$?”

Program snippet
Demand-driven query:
“What is the abstract state at $l'$?”
i.e. “What is the value of cell $l'$?”
Demand-driven query:
“What is the abstract state at \( l' \)?”
i.e. “What is the value of cell \( l' \)?”
Demand-driven query:

“What is the abstract state at \( l' \)?”

i.e. “What is the value of cell \( l' \)?”

\[
 l' = \underbrace{[s]}_{\#} l
\]
Demand-driven query:
“What is the abstract state at \( l' \)?”
i.e. “What is the value of cell \( l' \)?”

\[
\begin{align*}
\varphi_l &\quad l \rightarrow l' \\
\begin{array}{c}
\downarrow \\
\varphi_l
\end{array} &\quad s \\
\downarrow &\quad \# \\
\begin{array}{c}
\downarrow \quad ? \\
\varphi_l
\end{array} &\quad l' = \llbracket s \rrbracket \# l
\end{align*}
\]
Demand-driven query:

“What is the abstract state at \( l' \)?”

i.e. “What is the value of cell \( l' \)?”

\[
\begin{align*}
\l' &= \left[l \quad \varphi_l \quad l \rightarrow l' \quad s \right]^\# \\
&= \left[\varphi_l \quad l \rightarrow l' \quad s \right]^\# \\
&= \left[l \quad \varphi_l \quad l \rightarrow l' \quad s \right]^\#
\end{align*}
\]
Demand-driven query:
"What is the abstract state at $l'$?"

i.e. "What is the value of cell $l'$?"

Program snippet
Incremental Change Propagation

\[ \phi_l \xrightarrow{l \rightarrow l'} \phi_{l'} \]

\[ \llbracket \cdot \rrbracket^\# \]

\[ s \]
Program edit:
Modify statement $s$ to $s'$
Incremental Change Propagation

Program edit:
Modify statement $s$ to $s'$
i.e. “Write $s'$ to the cell named $l \rightarrow l'$”
Program edit:
Modify statement $s$ to $s'$
i.e. “Write $s'$ to the cell named $l \rightarrow l'$”
Incremental Change Propagation

Program edit:
Modify statement \( s \) to \( s' \)
i.e. “Write \( s' \) to the cell named \( l \rightarrow l' \)”

Eagerly discard forwards-reachable analysis results
Program edit: Modify statement $s$ to $s'$

i.e. “Write $s'$ to the cell named $l \rightarrow l'$”

Eagerly discard forwards-reachable analysis results
Incremental Change Propagation

Program edit: Modify statement $s$ to $s'$

i.e. “Write $s'$ to the cell named $l \rightarrow l'$”

Eagerly discard forwards-reachable analysis results

… recursively
Program edit: Modify statement $s$ to $s'$

i.e. “Write $s'$ to the cell named $l \rightarrow l'$”

Subsequent queries only recompute those analysis results potentially affected by the edit!
Cyclic Control Flow

Program snippet

Corresponding DAIG snippet

\[ l \rightarrow s_1 \rightarrow s_2 \rightarrow l' \]
Cyclic Control Flow

Program snippet

Corresponding DAIG snippet

\[ l_0 \]

?
Cyclic Control Flow

Program snippet

Corresponding DAIG snippet

\[
\begin{align*}
\text{l} & \rightarrow \text{l}' \\
\text{s}_1 & \rightarrow \text{l}' \\
\text{l}' & \rightarrow \text{l}
\end{align*}
\]
Cyclic Control Flow

Program snippet

Corresponding DAIG snippet
Cyclic Control Flow

Program snippet

\[
\begin{align*}
l & \rightarrow l' \\
\cdots & \\
& \\
l & \rightarrow l' \\
\end{align*}
\]

\[
\begin{align*}
l_0 & \rightarrow l_1 \\
\cdots & \\
l_0 & \rightarrow l_1 \\
\end{align*}
\]

Corresponding DAIG snippet

\[
\begin{align*}
l_0 & \rightarrow l_1 \\
l_0 & \rightarrow l_1 \\
\cdots & \\
\end{align*}
\]

\[
\begin{align*}
l_0 & \rightarrow l_1 \\
l_0 & \rightarrow l_1 \\
\cdots & \\
\end{align*}
\]

\[
\begin{align*}
l_0 & \rightarrow l_1 \\
l_0 & \rightarrow l_1 \\
\cdots & \\
\end{align*}
\]

\[
\begin{align*}
l_0 & \rightarrow l_1 \\
l_0 & \rightarrow l_1 \\
\cdots & \\
\end{align*}
\]

\[
\begin{align*}
l_0 & \rightarrow l_1 \\
l_0 & \rightarrow l_1 \\
\cdots & \\
\end{align*}
\]

\[
\begin{align*}
l_0 & \rightarrow l_1 \\
l_0 & \rightarrow l_1 \\
\cdots & \\
\end{align*}
\]

\[
\begin{align*}
l_0 & \rightarrow l_1 \\
l_0 & \rightarrow l_1 \\
\cdots & \\
\end{align*}
\]

\[
\begin{align*}
l_0 & \rightarrow l_1 \\
l_0 & \rightarrow l_1 \\
\cdots & \\
\end{align*}
\]

\[
\begin{align*}
l_0 & \rightarrow l_1 \\
l_0 & \rightarrow l_1 \\
\cdots & \\
\end{align*}
\]

\[
\begin{align*}
l_0 & \rightarrow l_1 \\
l_0 & \rightarrow l_1 \\
\cdots & \\
\end{align*}
\]

\[
\begin{align*}
l_0 & \rightarrow l_1 \\
l_0 & \rightarrow l_1 \\
\cdots & \\
\end{align*}
\]

\[
\begin{align*}
l_0 & \rightarrow l_1 \\
l_0 & \rightarrow l_1 \\
\cdots & \\
\end{align*}
\]

\[
\begin{align*}
l_0 & \rightarrow l_1 \\
l_0 & \rightarrow l_1 \\
\cdots & \\
\end{align*}
\]

\[
\begin{align*}
l_0 & \rightarrow l_1 \\
l_0 & \rightarrow l_1 \\
\cdots & \\
\end{align*}
\]
Cyclic Control Flow

Program snippet

Corresponding DAIG snippet

\[ s_1 \rightarrow l \rightarrow l' \rightarrow s_2 \]

\[ l_0 \rightarrow l_{0,1} \rightarrow l_1 \rightarrow l^* \]
Cyclic Control Flow

Program snippet

Corresponding DAIG snippet
DAIG Fixed-point Computation

Program snippet
DAIG Fixed-point Computation

Program snippet

Demand-driven query:
“What is the fixed-point abstract state at \( l \) ?”
DAIG Fixed-point Computation

Program snippet

Demand-driven query:
“What is the fixed-point abstract state at \( l \)?”
i.e. “What is the value of cell \( l^* \)?”
DAIG Fixed-point Computation

Demand-driven query:
“What is the fixed-point abstract state at $l$?”
i.e. “What is the value of cell $l^*$?”

Compute dependencies as normal…
DAIG Fixed-point Computation

Program snippet

Demand-driven query:
“What is the fixed-point abstract state at $l$?”
i.e. “What is the value of cell $l^*$?”

Compute dependencies as normal…
DAIG Fixed-point Computation

Program snippet

Demand-driven query:
“What is the fixed-point abstract state at $l$?”
i.e. “What is the value of cell $l^*$?”

Compute dependencies as normal…
DAIG Fixed-point Computation

Demand-driven query:
“What is the fixed-point abstract state at $l$?”
i.e. “What is the value of cell $l^*$?”

Compute dependencies as normal…
DAIG Fixed-point Computation

Program snippet

Demand-driven query:
“What is the fixed-point abstract state at $l$?”
i.e. “What is the value of cell $l^*$?”

Compute dependencies as normal…
DAIG Fixed-point Computation

Demand-driven query:
"What is the fixed-point abstract state at $l$?"

i.e. "What is the value of cell $l^*$?"

If the zeroth and first abstract iterates are equal (i.e. $\phi_{l}^0 = \phi_{l}^1$) then that’s the fixed point; write to $l^*$ and return.
DAIG Fixed-point Computation

Program snippet

Demand-driven query:
“What is the fixed-point abstract state at \( l \) ?”
i.e. “What is the value of cell \( l^* \) ?”

If the zeroth and first abstract iterates are equal (i.e. \( \varphi^0_l = \varphi^1_l \)) then that’s the fixed point; write to \( l^* \) and return.
DAIG Fixed-point Computation

Program snippet

Demand-driven query:
“What is the fixed-point abstract state at \( l \) ?”

Otherwise, unroll the DAIG’s loop representation, re-query, and continue.
Demand-driven query: “What is the fixed-point abstract state at $l$?”

Otherwise, unroll the DAIG’s loop representation, re-query, and continue.
Demand-driven query:
“What is the fixed-point abstract state at \( l \)?”

Widen’s “ascending-chains-converge” property ensures that unrolling is finite!

Otherwise, unroll the DAIG’s loop representation, re-query, and continue.
A DAIG is an *explicit* representation of a partially-evaluated abstract interpretation.
Query evaluation == (small-step) operational semantics
DAIG Formalism

Query evaluation == (small-step) operational semantics

Given an initial DAIG*...
Given an initial DAIG*... ... a query for the value named $n$ yields $v$...
Given an initial DAIG*...
... a query for the value named $n$ yields $v$...
... and an updated DAIG*.

* this elides some details of the actual semantics
Query evaluation == (small-step) operational semantics

Given an initial DAIG*...

... a query for the value named $n$ yields $v$...

... and an updated DAIG*.

Edit handling/change propagation is also a small-step operational semantics

* this elides some details of the actual semantics
Query evaluation == (small-step) operational semantics

\[ \vdash n \Rightarrow v ; \]

Given an initial DAIG*...

... a query for the value named \( n \) yields \( v \)...

... and an updated DAIG*.

Edit handling/change propagation is also a small-step operational semantics

Given an initial DAIG...

* this elides some details of the actual semantics
DAIG Formalism

Query evaluation == (small-step) operational semantics

Given an initial DAIG*…

... a query for the value named \( n \) yields \( v \)…

... and an updated DAIG*.

Edit handling/change propagation is also a small-step operational semantics

Given an initial DAIG...

... an edit that writes value \( v \) to cell \( n \)…

* this elides some details of the actual semantics
DAIG Formalism

Query evaluation == (small-step) operational semantics

\[ \vdash n \Rightarrow v \; ; \]

Given an initial DAIG*...

... a query for the value named \( n \) yields \( v \)...

... and an updated DAIG*.

Edit handling/change propagation is also a small-step operational semantics

\[ \vdash n \Leftarrow v \; ; \]

Given an initial DAIG...

... an edit that writes value \( v \) to cell \( n \) ...

... yields an updated DAIG.

* this elides some details of the actual semantics
Termination
Termination

If this initial DAIG is a valid abstract interpretation state for some program...
If this initial DAIG is a valid abstract interpretation state for some program...

... then a query for any $n$ therein ...
If this initial DAIG is a valid abstract interpretation state for some program...

... then a query for any $n$ therein ...

... will terminate with some value $v$ and an updated DAIG.
From-Scratch Consistency

\[ \vdash n \Rightarrow v ; \]
From-Scratch Consistency

If the current DAIG is a valid abstract interpretation state for some program...
If the current DAIG is a valid abstract interpretation state for some program...

... and $n$ is the name of the abstract state at a program location $l$ ...
From-Scratch Consistency

$\Gamma \vdash n \Rightarrow v ;$  

If the current DAIG is a valid abstract interpretation state for some program...

... and $n$ is the name of the abstract state at a program location $l$ ...

... then $v$ is precisely the same value that a batch abstract interpreter would compute at $l$.  


From-Scratch Consistency

If the current DAIG is a valid abstract interpretation state for some program...

... and \( n \) is the name of the abstract state at a program location \( l \) ...

... then \( \nu \) is precisely the same value that a batch abstract interpreter would compute at \( l \).

No loss of precision due to incrementality/demand!
From-Scratch Consistency

If the current DAIG is a valid abstract interpretation state for some program...

... and $n$ is the name of the abstract state at a program location $l$ ...

... then $v$ is precisely the same value that a batch abstract interpreter would compute at $l$.

No loss of precision due to incrementality/demand!

Corollary: DAIG query results are **sound**.
Expressivity

Does a DAIG-based analysis framework support rich analysis domains that cannot be handled by existing incremental and/or demand-driven frameworks?
Expressivity

Does a DAIG-based analysis framework support rich analysis domains that cannot be handled by existing incremental and/or demand-driven frameworks?

Prototype Implementation: [github.com/cuplv/dai](https://github.com/cuplv/dai)
- ∼2500 lines of OCaml code
- Parametric in a statement language and abstract domain
Expressivity

Does a DAIG-based analysis framework support rich analysis domains that cannot be handled by existing incremental and/or demand-driven frameworks?

Prototype Implementation: [github.com/cuplv/dai](https://github.com/cuplv/dai)
- ~2500 lines of OCaml code
- Parametric in a statement language and abstract domain

```ocaml
module type Domain = sig

    type t

    val init : t
    val interpret : stmt -> t -> t
    val implies : t -> t -> bool
    val join : t -> t -> t
    val widen : t -> t -> t

    (* elided: equal, hash, etc. *)
end
```
Expressivity

Does a DAIG-based analysis framework support rich analysis domains that cannot be handled by existing incremental and/or demand-driven frameworks?

Prototype Implementation: github.com/cuplv/dai
- ~2500 lines of OCaml code
- Parametric in a statement language and abstract domain

abstract states $\Sigma^#$
initial abstract state $\phi_0$
transfer function $\llbracket \cdot \rrbracket^#$
partial order $\sqsubseteq$
join $\sqcup$
widening $\nabla$

```
module type Domain = sig
  type t
  val init : t
  val interpret : stmt -> t -> t
  val implies : t -> t -> bool
  val join : t -> t -> t
  val widen : t -> t -> t
  (* elided: equal, hash, etc. *)
end
```
Expressivity

Does a DAIG-based analysis framework support rich analysis domains that cannot be handled by existing incremental and/or demand-driven frameworks?

Prototype Implementation: [github.com/cuplv/dai](https://github.com/cuplv/dai)
- ~2500 lines of OCaml code
- Parametric in a statement language and abstract domain

Module type `Domain = sig`:

```ocaml
module type Domain = sig
  type t
  val init : t
  val interpret : stmt -> t -> t
  val implies : t -> t -> bool
  val join : t -> t -> t
  val widen : t -> t -> t
  (* elided: equal, hash, etc. *)
end
```

Domain implementer doesn’t need to reason about incrementality or demand!
Expressivity

Do DAIGs support rich abstract domains that cannot be handled by existing incremental and/or demand-driven frameworks?
Expressivity

Do DAIGs support rich abstract domains that cannot be handled by existing incremental and/or demand-driven frameworks?

Interval Analysis

- Abstract values $[x, y]$ model integers $\{i | x \leq i \leq y\}$
- Used to verify array accesses in-bounds in a JS data structure library.
Expressivity

Do DAIGs support rich abstract domains that cannot be handled by existing incremental and/or demand-driven frameworks?

**Interval Analysis**
- Abstract values $[x, y]$ model integers $\{i | x \leq i \leq y\}$
- Used to verify array accesses in-bounds in a JS data structure library.

**Shape Analysis**
- Separation logic formulae over linked-list-segment primitive $lseg(\hat{x}, \hat{y})$
- Used to verify memory safety of linked-list `append`, `reverse`, etc.
Expressivity

Do DAIGs support rich abstract domains that cannot be handled by existing incremental and/or demand-driven frameworks?

**Interval Analysis**
- Abstract values $[x, y]$ model integers $\{i \mid x \leq i \leq y\}$
- Used to verify array accesses in-bounds in a JS data structure library.

**Shape Analysis**
- Separation logic formulae over linked-list-segment primitive $\text{lseg}(\hat{x}, \hat{y})$
- Used to verify memory safety of linked-list $\text{append}$, $\text{reverse}$, etc.

**Octagon Analysis**
- Invariants of the form $\pm x \pm y \leq c$
- Used in scalability experiments
Expressivity

Do DAIGs support rich abstract domains that cannot be handled by existing incremental and/or demand-driven frameworks?

**Interval Analysis**
- Abstract values \([x, y]\) model integers \(\{i \mid x \leq i \leq y\}\)
- Used to verify array accesses in-bounds in a JS data structure library.

**Shape Analysis**
- Separation logic formulae over linked-list-segment primitive \(\text{lseg}(\hat{x}, \hat{y})\)
- Used to verify memory safety of linked-list \text{append}, \text{reverse}, etc.

**Octagon Analysis**
- Invariants of the form \(\pm x \pm y \leq c\)
- Used in scalability experiments

**Intervals & Octagons built with APRON** — optimized open-source numerical domains in C
For these rich analysis domains, what degree of performance improvement can be obtained by performing incremental and/or demand-driven analysis?
Scalability

For these rich analysis domains, what degree of performance improvement can be obtained by performing incremental and/or demand-driven analysis?

“Edit” = add a random statement, conditional, or loop at a random program location
Scalability

For these rich analysis domains, what degree of performance improvement can be obtained by performing incremental and/or demand-driven analysis?

Initial program: skip

“Edit” = add a random statement, conditional, or loop at a random program location

~5k LOC
For these rich analysis domains, what degree of performance improvement can be obtained by performing incremental and/or demand-driven analysis?
Scalability
Scalability

- Batch
- Demand-Driven
- Incremental
Scalability
The *combination* of incrementality and demand consistently obtains lower latencies than either incrementality or demand alone.
The combination of incrementality and demand consistently obtains lower latencies than either incrementality or demand alone.

P95: 1.2s; 6.3s; 7.9s; 36.2s
Thanks for watching!

Check out our paper or come chat at the Q&A for more details.

Conclusion:

- Batch whole-program analysis is too costly to support real-time developer interaction, but existing incremental and demand-driven analyses are often limited in expressivity or granularity.
- By leveraging graph-based incremental computation techniques, we define an engine for incremental and demand-driven evaluation of *arbitrary* abstract interpreters (and prove it sound & from-scratch consistent).

Benno Stein
University of Colorado Boulder
benno.stein@colorado.edu