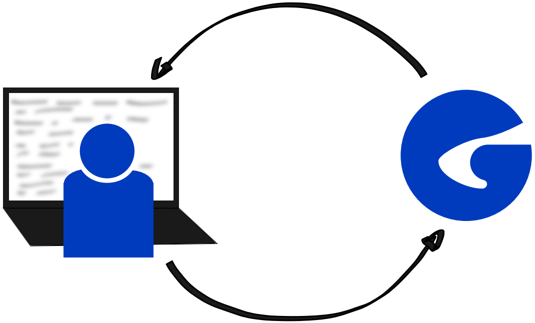
Towards Interactive Abstract Interpretation for Multithreaded Programs

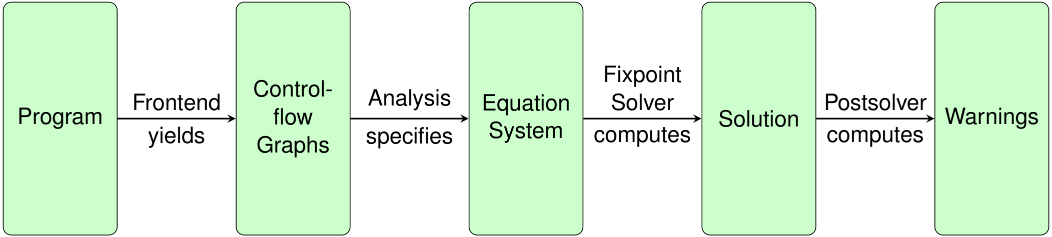
Michael Schwarz Helmut Seidl
+
Julian Erhard Karoliine Holter Simmo Saan
Sarah Tilscher Vesal Vojdani

Schloss Dagstuhl, October 2025

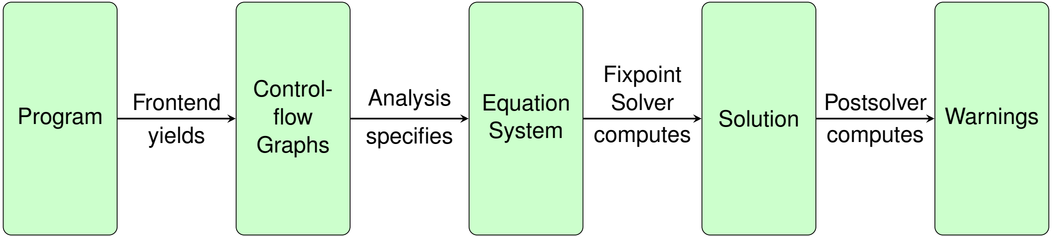
Provide Abstract Interpretation During Development



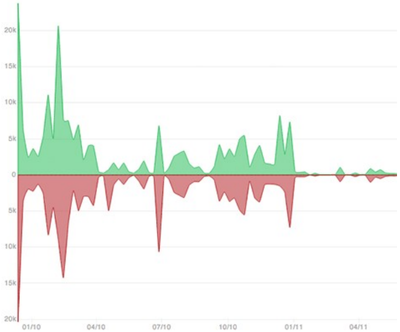
Abstract interpretation: Expensive Fixpoint Computation



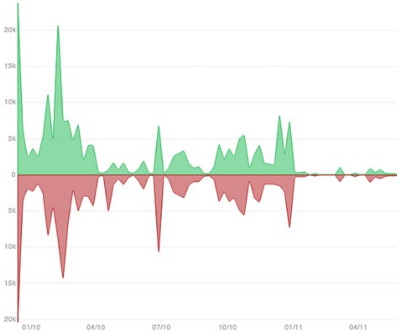
Abstract interpretation: Expensive Fixpoint Computation



Observation: Software development is incremental

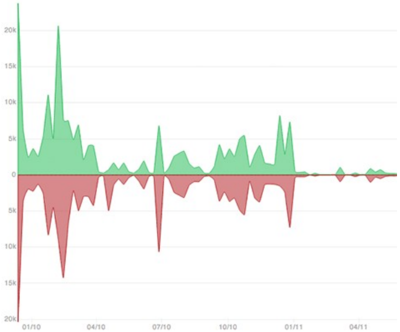


Observation: Software development is incremental



- Software development is **incremental**

Observation: Software development is incremental



- Software development is **incremental**
- Single commits usually make **small changes**

Idea: Static analysis should be incremental, too!

Re-analyze only the increments

Idea: Static analysis should be incremental, too!

Re-analyze only the increments

... and their impact

Idea: Static analysis should be incremental, too!

Re-analyze only the increments

... and their impact

Required: semantic dependencies

Idea: Static analysis should be incremental, too!

Re-analyze only the increments

... and their impact

Required: semantic dependencies

The **top-down solver** TD already tracks dependencies

Idea: Static analysis should be incremental, too!

Re-analyze only the increments

... and their impact

Required: semantic dependencies

The **top-down solver** TD already tracks dependencies

TD maintains dependencies and stable unknowns.

TD tracks

TD maintains dependencies and stable unknowns.

TD tracks

- set of `stable` unknowns

TD maintains dependencies and stable unknowns.

TD tracks

- set of **`stable`** unknowns
- set of **`called`** unknowns – those are currently active

TD maintains dependencies and stable unknowns.

TD tracks

- set of `stable` unknowns
- set of `called` unknowns – those are currently active
- influence relationship between unknowns

TD maintains dependencies and stable unknowns.

TD tracks

- set of `stable` unknowns
- set of `called` unknowns – those are currently active
- influence relationship between unknowns
- mapping from unknowns to abstract values

TD maintains dependencies and stable unknowns.

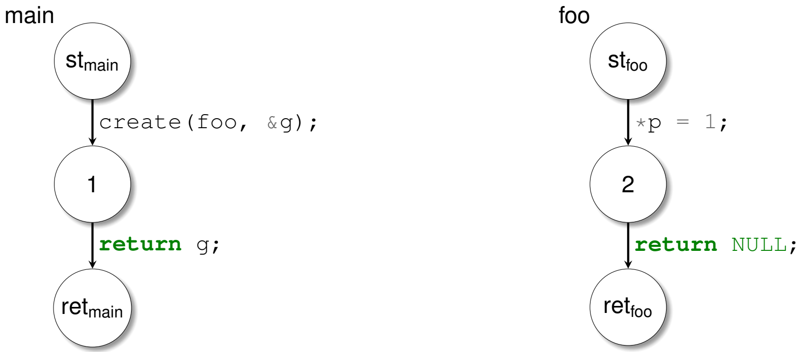
TD tracks

- set of `stable` unknowns
- set of `called` unknowns – those are currently active
- influence relationship between unknowns
- mapping from unknowns to abstract values
- when unknown changes, it is *destabilized*
  — remove all unknowns from `stable` that depend on it and are not `called`.

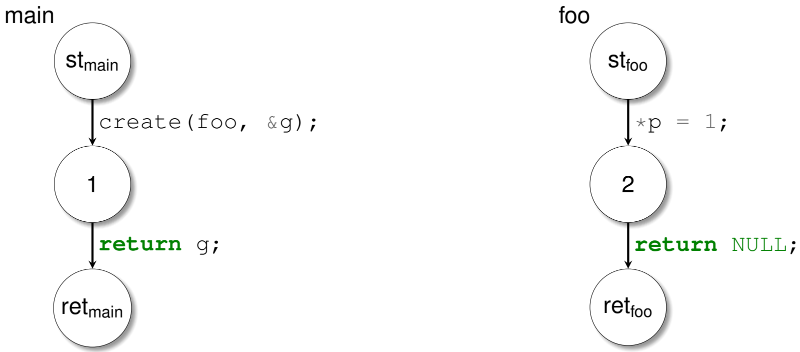
Example program

```
int g = 0;
int main(){
        create(foo, &g);
        return g;
}
void* foo(int *p){
        *p = 1;
        return NULL;
}
```

Example program

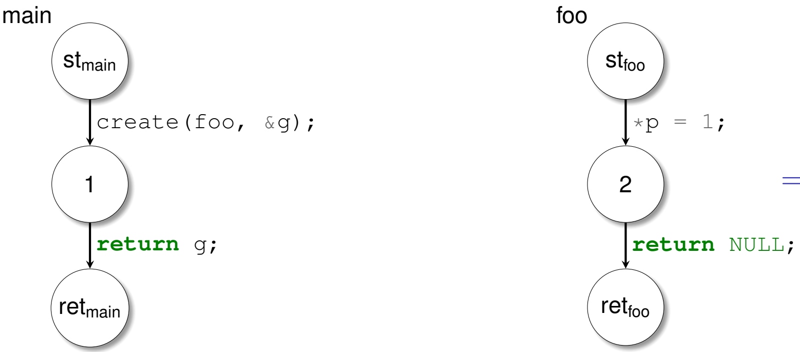


Example program



- Treat variables shared between threads flow-insensitively:

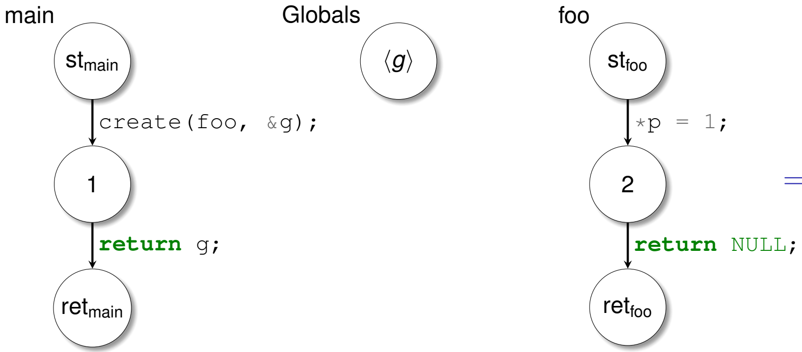
Example program



- Treat variables shared between threads flow-insensitively:

$\Longrightarrow$ Add unknown $\langle g \rangle$ to collect abstraction of all values `g` may ever have

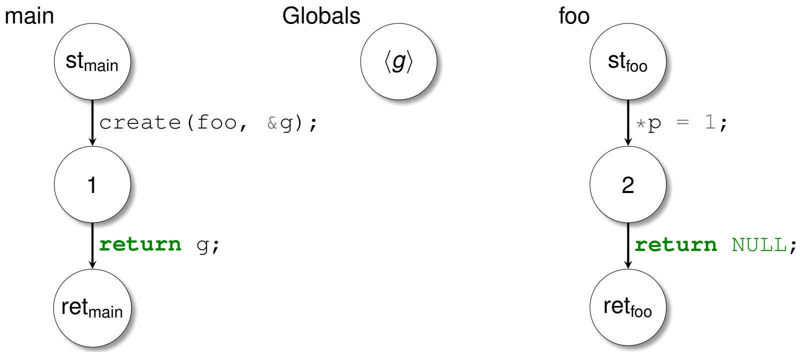
Example program



- Treat variables shared between threads flow-insensitively:

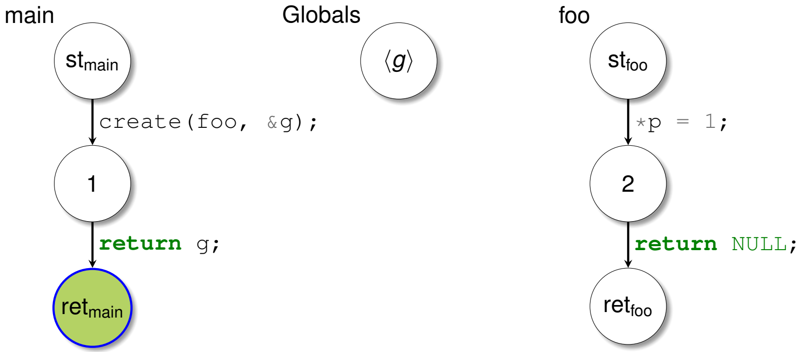
$\Longrightarrow$ Add unknown $\langle g \rangle$ to collect abstraction of all values `g` may ever have

Example Run of Top-Down Solver



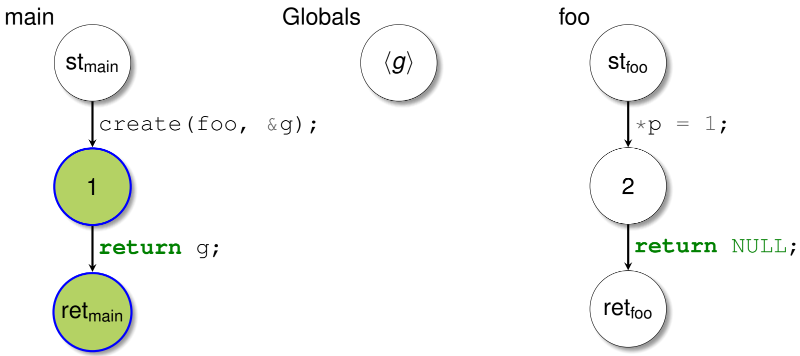
- Call to **`solve`** $\mathrm{ret}_{\mathrm{main}}$
- recursively computes solution
- **`stable`**
- **`called`**

Example Run of Top-Down Solver



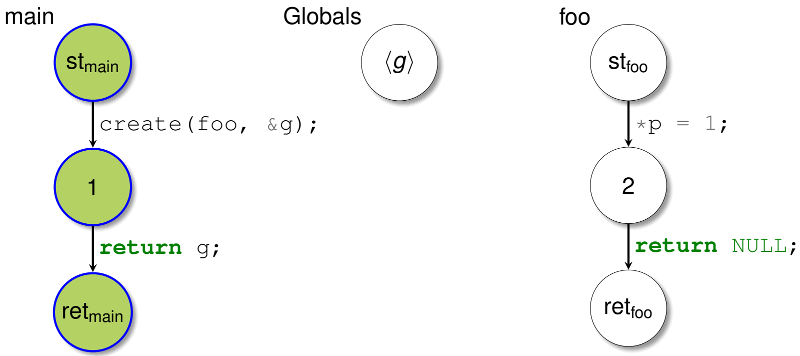
- Call to **solve** ret_{main}
- recursively computes solution
- **stable**
- **called**

Example Run of Top-Down Solver



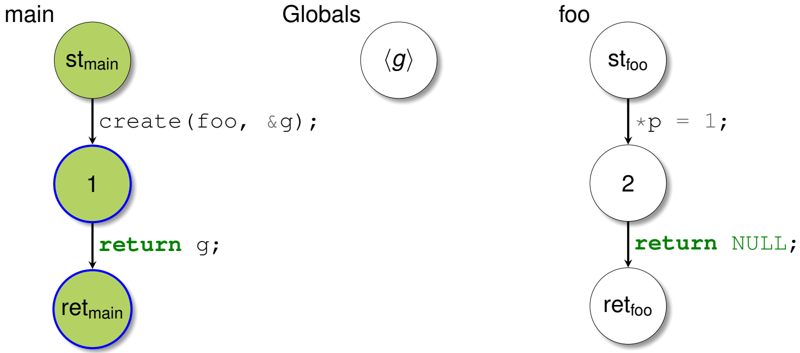
- Call to **`solve`** $\mathrm{ret}_{\mathrm{main}}$
- recursively computes solution
- **`stable`**
- **`called`**

Example Run of Top-Down Solver



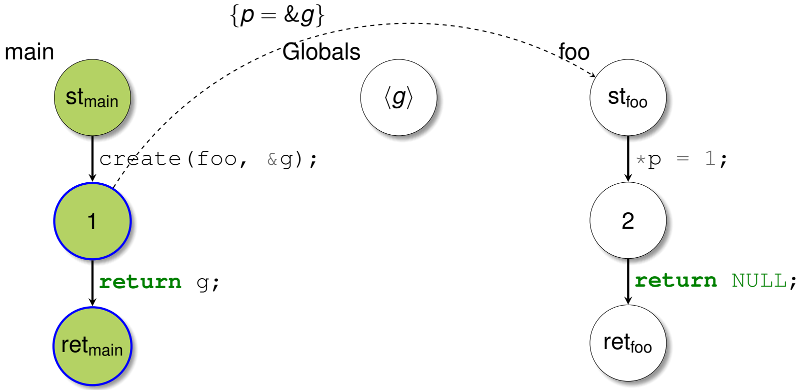
- Call to **`solve`** ret$_{main}$
- recursively computes solution
- **`stable`**
- **`called`**

Example Run of Top-Down Solver



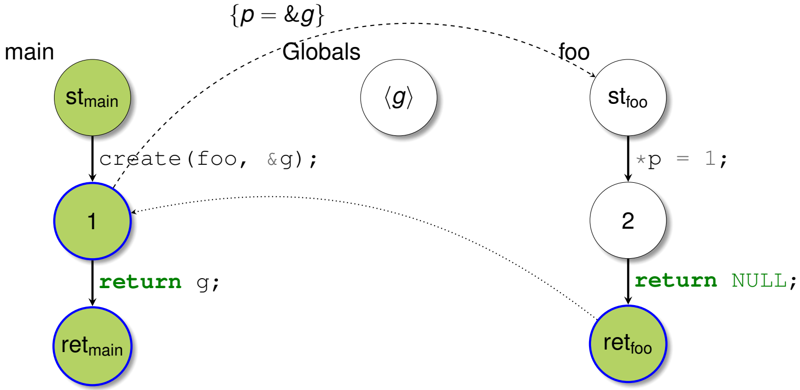
- Call to **`solve`** ret_{main}
- recursively computes solution
- **`stable`**
- **`called`**

Example Run of Top-Down Solver



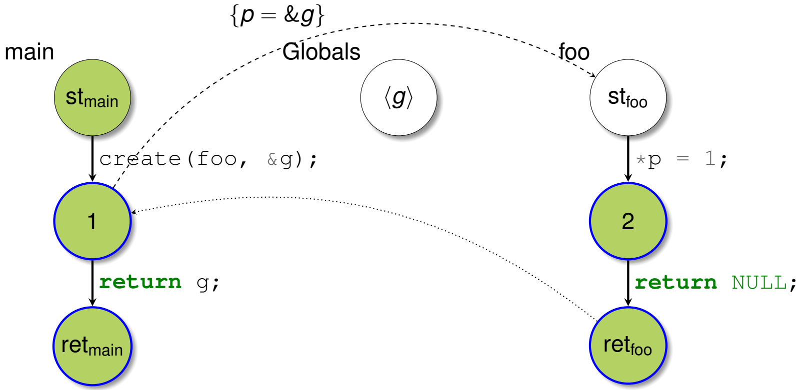
- Call to **solve** ret_{main}
- recursively computes solution
- **stable**
- **called**

Example Run of Top-Down Solver



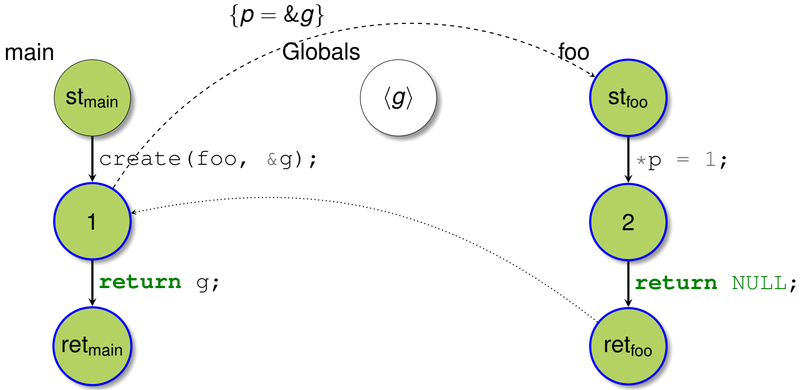
- Call to **solve** $\mathrm{ret}_{\mathrm{main}}$
- recursively computes solution
- **stable**
- **called**

Example Run of Top-Down Solver



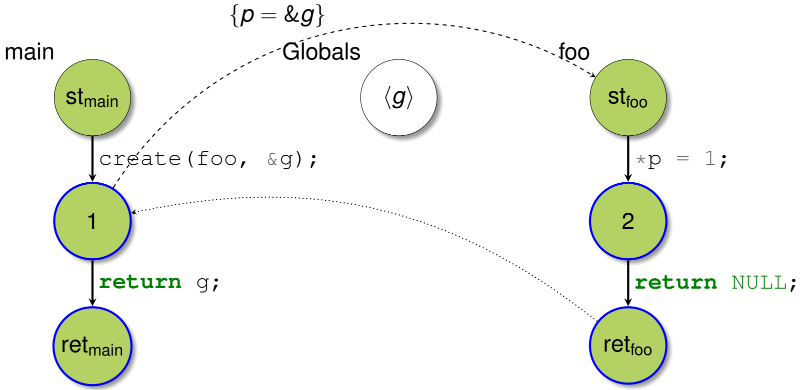
- Call to **solve** ret_{main}
- recursively computes solution
- **stable**
- **called**

Example Run of Top-Down Solver



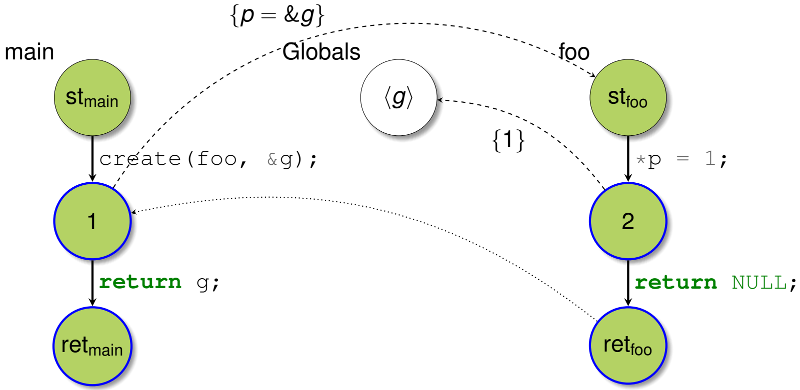
- Call to **solve** $\mathrm{ret}_{\mathrm{main}}$
- recursively computes solution
- **stable**
- **called**

Example Run of Top-Down Solver



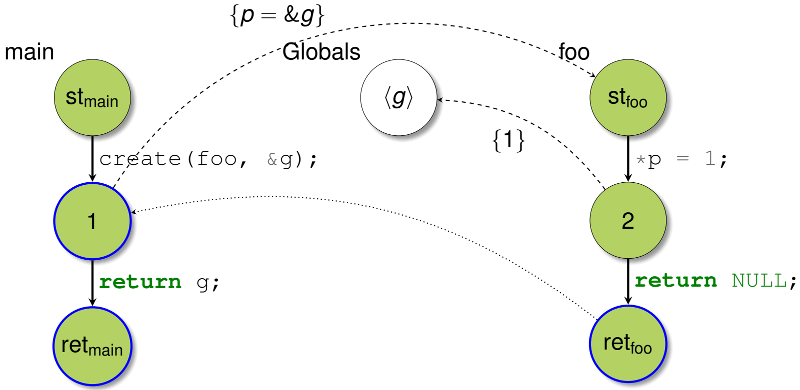
- Call to **`solve`** $\mathrm{ret}_{\mathrm{main}}$
- recursively computes solution
- **`stable`**
- **`called`**

Example Run of Top-Down Solver



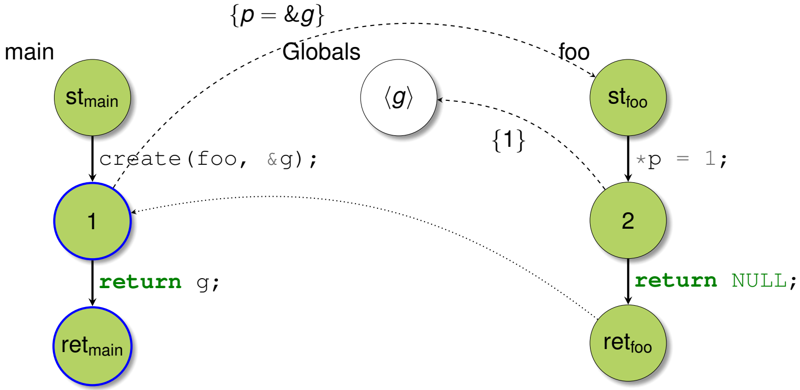
- Call to **solve** $\mathrm{ret}_{\mathrm{main}}$
- recursively computes solution
- **stable**
- **called**

Example Run of Top-Down Solver



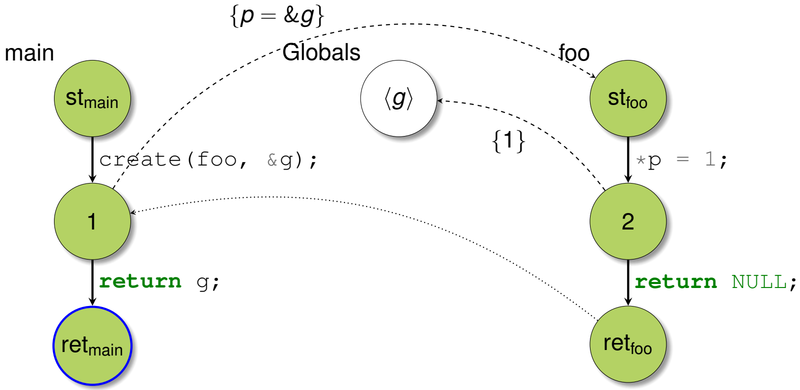
- Call to **solve** ret_{main}
- recursively computes solution
- **stable**
- **called**

Example Run of Top-Down Solver



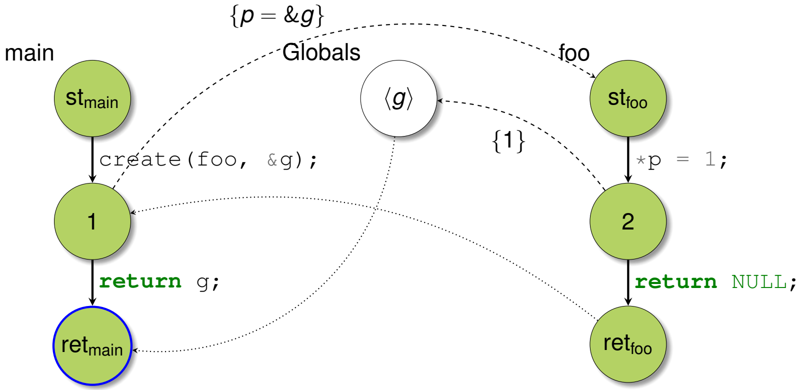
- Call to **`solve`** $\mathrm{ret}_{\mathrm{main}}$
- recursively computes solution
- **`stable`**
- **`called`**

Example Run of Top-Down Solver



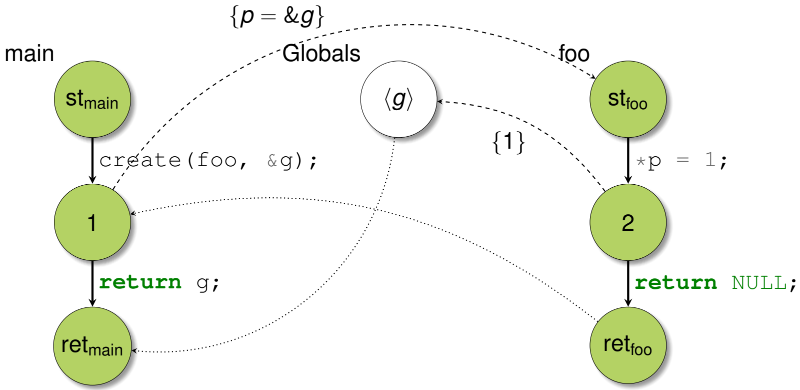
- Call to **solve** ret_{main}
- recursively computes solution
- **stable**
- **called**

Example Run of Top-Down Solver



- Call to **`solve`** $\mathrm{ret}_{\mathrm{main}}$
- recursively computes solution
- **`stable`**
- **`called`**

Example Run of Top-Down Solver



- Call to **solve** $\mathrm{ret}_{\mathrm{main}}$
- recursively computes solution
- **stable**
- **called**

TD is versatile.

TD supports

TD is versatile.

TD supports

- **dynamic dependencies** between unknowns
- demand-driven analysis
- widening/narrowing

TD is versatile.

TD supports

- **dynamic dependencies** between unknowns
- demand-driven analysis
- widening/narrowing
- **mixed flow-sensivitity**, i.e., may collect **flow-insensitive information** while performing a **flow-sensitive** analysis otherwise

Change in example program

```
int g = 0;
int main(){
        create(foo, &g);
        return g;
}
void* foo(int *p){
        *p = 1;
        return NULL;
}
```

Change in example program

```
int g = 0;
int main(){
        create(foo, &g);
        return g;
}
void* foo(int *p){
        *p = 1;
        return NULL;
}
```

$\Longrightarrow$

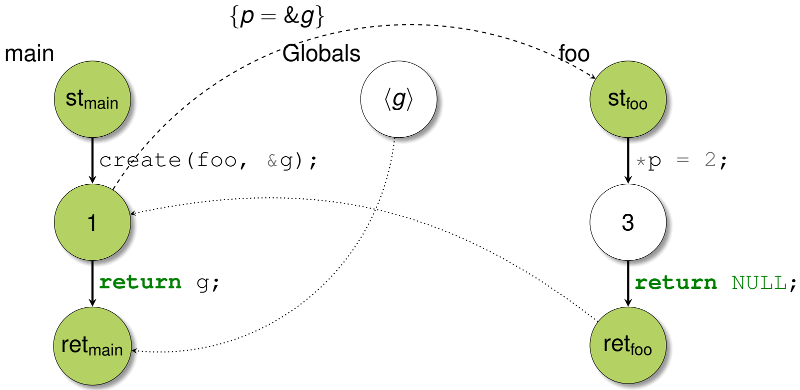
Change in example program

```c
int g = 0;
int main(){
        create(foo, &g);
        return g;
}
void* foo(int *p){
        *p = 1;
        return NULL;
}
```

$\Longrightarrow$

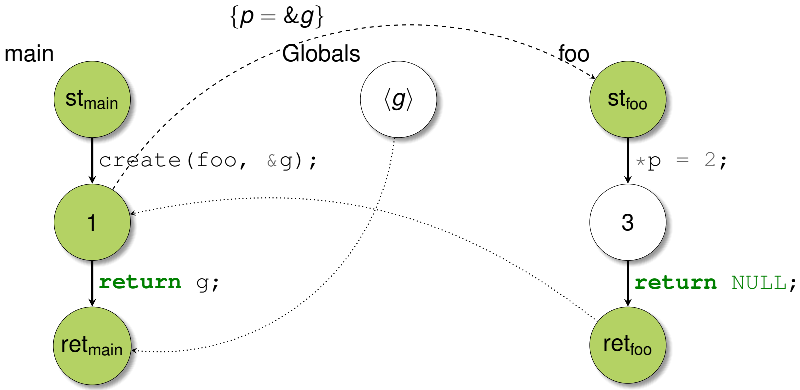
```c
int g = 0;
int main(){
        create(foo, &g);
        return g;
}
void* foo(int *p){
        *p = 2;
        return NULL;
}
```

Incremental Abstract Interpretation



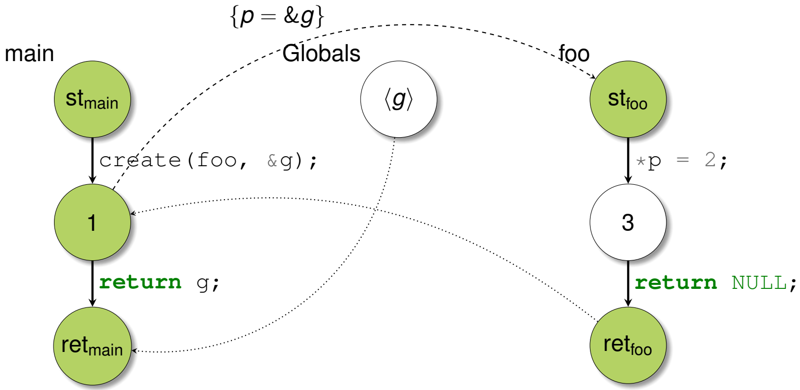
1. Load old results

Incremental Abstract Interpretation



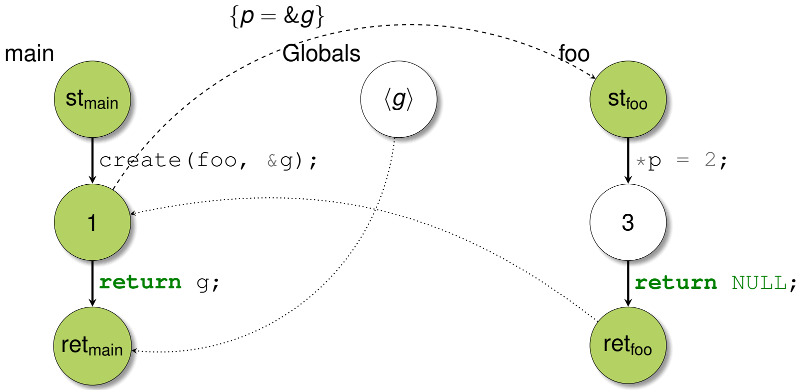
1. Load old results
2. Identify changed functions

Incremental Abstract Interpretation



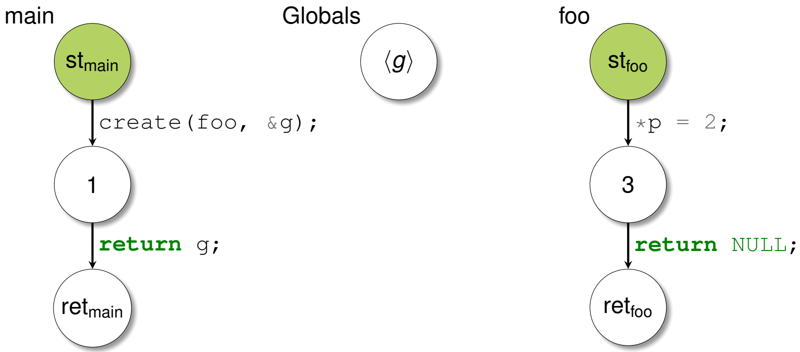
1. Load old results
2. Identify changed functions
   $\Longrightarrow$ `foo`

Incremental Abstract Interpretation



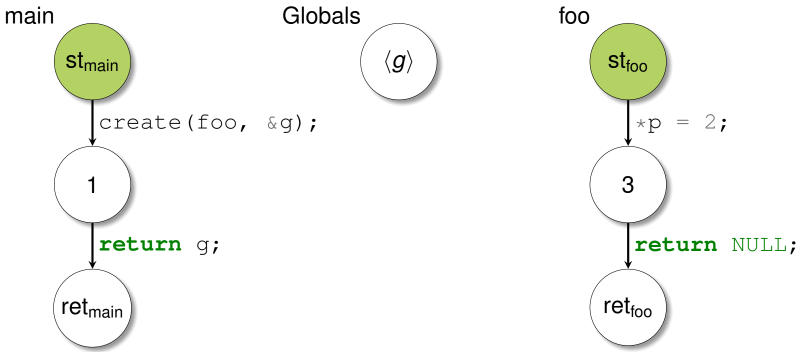
1. Load old results
2. Identify changed functions
   $\Longrightarrow$ `foo`
3. Destabilize return nodes of changed functions

Incremental Abstract Interpretation



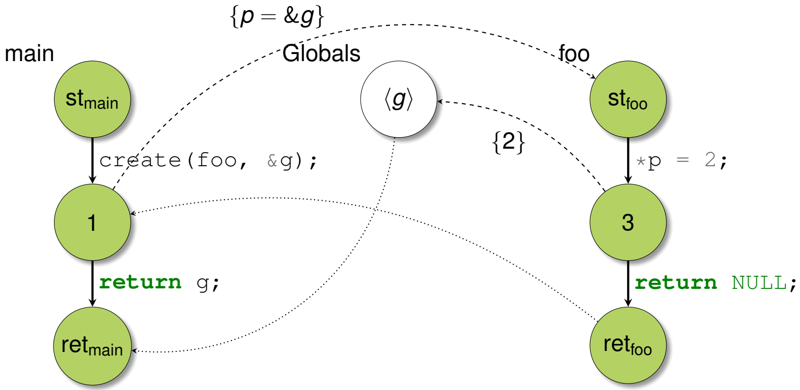
1. Load old results
2. Identify changed functions
   $\Longrightarrow$ `foo`
3. Destabilize return nodes of changed functions

Incremental Abstract Interpretation



1. Load old results
2. Identify changed functions
   $\Longrightarrow$ `foo`
3. Destabilize return nodes of changed functions
4. Call solve for ret_{main}

Incremental Abstract Interpretation



1. Load old results
2. Identify changed functions
   $\Longrightarrow$ `foo`
3. Destabilize return nodes of changed functions
4. Call solve for ret_{main}

Incremental Abstract Interpretation

- **Reluctant Destabilization**
- Incremental Warning Generation

- Update rules for globals

- IDE Integration

Effects of changes are sometimes local to the procedure.

```
int g = 0;
int main(){
        create(foo, &g);
        return g;
}
void* foo(int *p){
        *p = 1;
        return NULL;
}
```

Effects of changes are sometimes local to the procedure.

```
int g = 0;
int main(){
        create(foo, &g);
        return g;
}
void* foo(int *p){
        *p = 1;
        return NULL;
}
```

$\Longrightarrow$

Effects of changes are sometimes local to the procedure.

```c
int g = 0;
int main(){
        create(foo, &g);
        return g;
}
void* foo(int *p){
        *p = 1;
        return NULL;
}
```

$\Longrightarrow$

```c
int g = 0;
int main(){
        create(foo, &g);
        return g;
}
void* foo(int *p){
        *p = 1;
        printf("foo\n");
        return NULL;
}
```

Effects of changes are sometimes local to the procedure.

```c
int g = 0;
int main(){
        create(foo, &g);
        return g;
}
void* foo(int *p){
        *p = 1;
        return NULL;
}
```

$\Longrightarrow$

```c
int g = 0;
int main(){
        create(foo, &g);
        return g;
}
void* foo(int *p){
        *p = 1;
        printf("foo\n");
        return NULL;
}
```

$\Longrightarrow$ Change has no impact outside of `foo`!

Reluctant Destabilization

1. Load results

Reluctant Destabilization

1. Load results
2. Destabilize changed functions and start solver from there
   Non-local changes will destabilize end point of `main`.

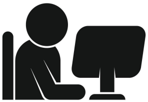
Reluctant Destabilization

1. Load results
2. Destabilize changed functions and start solver from there
   Non-local changes will destabilize end point of `main`.
3. Solve end point of `main`. If all changes were local, the end point of `main` is stable, so nothing to do.

Incremental Abstract Interpretation

- Reluctant Destabilization
- **Incremental Warning Generation**

- Update rules for globals



- IDE Integration

Observation: Warnings can be reused as well

```
int main(){
        int *i = get_pointer();
        return *i;
}
int* get_pointer(){
        return NULL;
}
```

Warning:
Possible NULL-dereference
in line 3.

Observation: Warnings can be reused as well

```
int main(){
        int *i = get_pointer();
        return *i;
}
int* get_pointer(){
        return NULL;
}
```

$\Longrightarrow$

Warning:
Possible NULL-dereference
in line 3.

Observation: Warnings can be reused as well

```c
int main(){
        int *i = get_pointer();
        return *i;
}
int* get_pointer(){
        return NULL;
}
```

Warning:
Possible NULL-dereference
in line 3.

$\Longrightarrow$

```c
int main(){
  int *i = get_pointer();
  return *i;
}
int* get_pointer(){
  printf("get_pointer\n");
  return NULL;
}
```

Warning:
Possible NULL-dereference
in line 3.

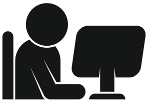
Incremental Warning Generation

- Keep track of unknowns never destabilized during incremental run
- Warnings generated at these unknowns can be reused

Incremental Abstract Interpretation

- Reluctant Destabilization
- Incremental Warning Generation

- **Update rules for globals**



- IDE Integration

Observation: Flow-insensitive invariants accumulate imprecision.

In our incremental analysis example, $\langle g \rangle$ received contributions:

Observation: Flow-insensitive invariants accumulate imprecision.

In our incremental analysis example, $\langle g \rangle$ received contributions:

- $\{0\}$, initial value in both versions

Observation: Flow-insensitive invariants accumulate imprecision.

In our incremental analysis example, $\langle g \rangle$ received contributions:

- $\{0\}$, initial value in both versions
- $\{1\}$, write by `foo` in first version

Observation: Flow-insensitive invariants accumulate imprecision.

In our incremental analysis example, $\langle g \rangle$ received contributions:

- $\{0\}$, initial value in both versions
- $\{1\}$, write by `foo` in first version
- $\{2\}$, write by `foo` in second version

Observation: Flow-insensitive invariants accumulate imprecision.

In our incremental analysis example, $\langle g \rangle$ received contributions:

- $\{0\}$, initial value in both versions
- $\{1\}$, write by `foo` in first version
- $\{2\}$, write by `foo` in second version

Thus, $\langle g \rangle$ will have the value $\{0, 1, 2\}$ in the incremental analysis.

Observation: Flow-insensitive invariants accumulate imprecision.

In our incremental analysis example, $\langle g \rangle$ received contributions:

- $\{0\}$, initial value in both versions
- $\{1\}$, write by `foo` in first version
- $\{2\}$, write by `foo` in second version

Thus, $\langle g \rangle$ will have the value $\{0, 1, 2\}$ in the incremental analysis.

$\Longrightarrow$ Flow-insensitive unknowns will contain artifacts from earlier programs.

Solutions

- Restarting
  - Restart flow-insensitive unknowns by setting them to $\perp$
  - Destabilize all unknowns that cause a side effect

Solutions

- Restarting
  - Restart flow-insensitive unknowns by setting them to $\bot$
  - Destabilize all unknowns that cause a side effect

- Update Rules and Abstract Garbage Collection

Solutions

- Restarting
  - Restart flow-insensitive unknowns by setting them to $\perp$
  - Destabilize all unknowns that cause a side effect

- Update Rules and Abstract Garbage Collection
  - Remember where contributions came from

Solutions

- Restarting
  - Restart flow-insensitive unknowns by setting them to $\perp$
  - Destabilize all unknowns that cause a side effect

- Update Rules and Abstract Garbage Collection
  - Remember where contributions came from
  - Δ upon re-evaluation

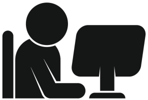
Solutions

- Restarting
  - Restart flow-insensitive unknowns by setting them to $\bot$
  - Destabilize all unknowns that cause a side effect

- Update Rules and Abstract Garbage Collection
  - Remember where contributions came from
  - Δ upon re-evaluation
  - Completely remove contributions from dead contexts

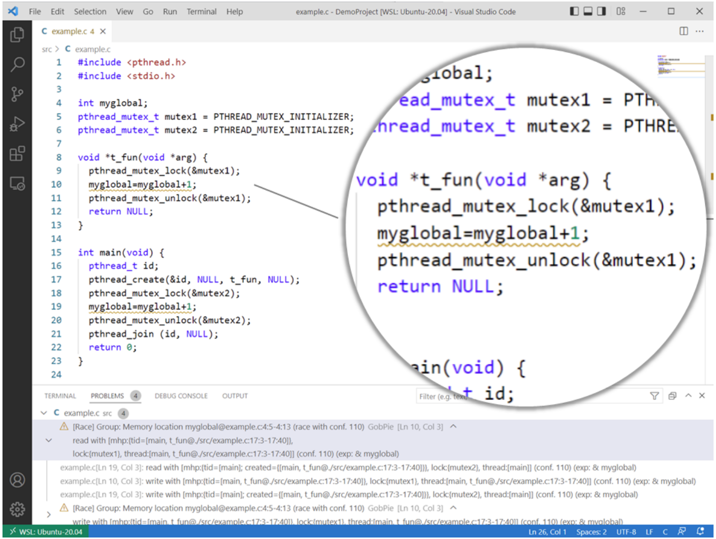
- Reluctant Destabilization
- Incremental Warning Generation

- Update rules for globals



- **IDE Integration**

Integrated into VS Code



Experiments

Analyze commits of ZSTANDARD compression algorithm (~22,300 LoC).

Experiments

Analyze commits of ZSTANDARD compression algorithm (~22,300 LoC).

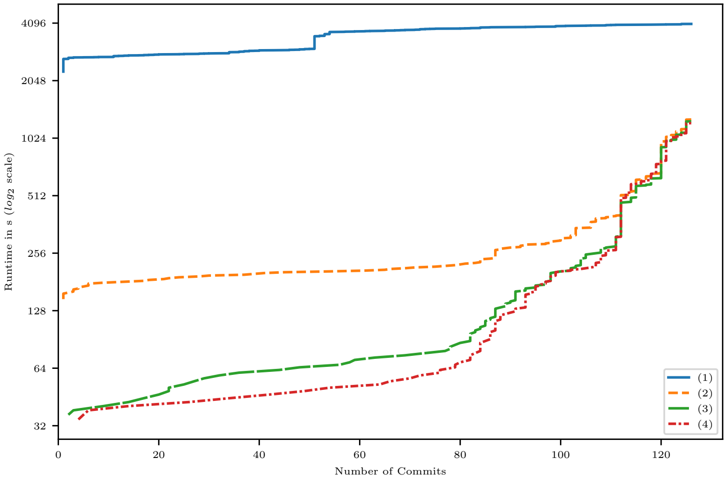
- Intervals & exclusion sets

Experiments

Analyze commits of ZSTANDARD compression algorithm (~22,300 LoC).

- Intervals & exclusion sets
- Accesses & data-race detection

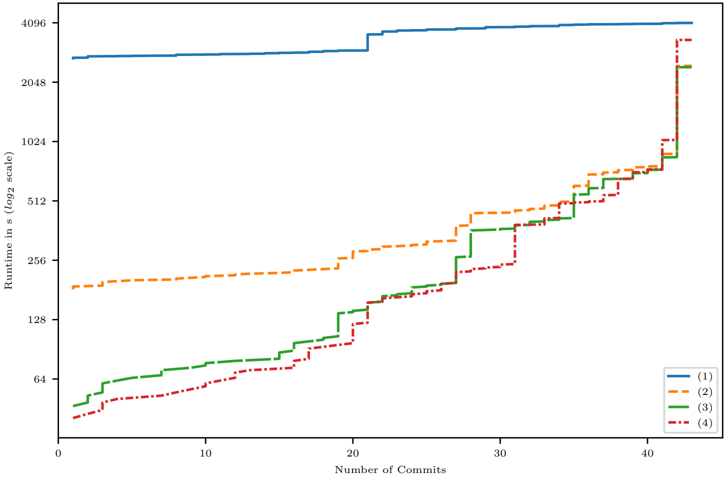
Results on ZSTANDARD



Analysis runtimes on commits with $\leq$ 50 lines of code changed. Configurations:

- non-incremental
- incremental
- + incremental warning generation
- + reluctant destabilization

Results on ZSTANDARD



Analysis runtimes on commits with $>$ 50 lines of code changed. Configurations:

- non-incremental
- incremental
- + incremental warning generation
- + reluctant destabilization

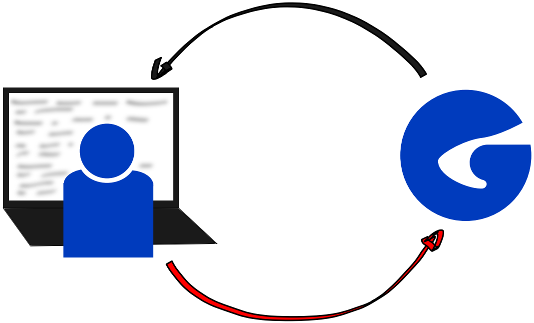
Our previous work

- [Seidl et al., Chrisfest'20]: Incremental Abstract Interpretation
- [Erhard et al., STTT'23]: Interactive Abstract Interpretation: Reanalyzing Multithreaded C Programs for Cheap
- [Stemmler et al., PLDI'25]: Taking out the Toxic Trash: Recovering Precision in Mixed Flow-Sensitive Static Analyses

GOBLINT

/goblint/analyzer

Vision: Interactive Abstract Interpretation During Development



From Incremental to Interactive

Consider not only changes to the input program

From Incremental to Interactive

Consider not only changes to the input program,
but also **user interaction**!

From Incremental to Interactive

Consider not only changes to the input program,
but also **user interaction**!

(one may think of it as changing the analysis specification)

Expert Developer Diane

Expert Developer Diane

- Coarse results immediately, refine in the background
- While fixing the most salient warning, the analysis may be able to show less salient warnings spurious

Expert Developer Diane

- Surface semantic information

Expert Developer Diane

- Surface semantic information
  - complete call-graph

Expert Developer Diane

- Surface semantic information
  - complete call-graph
  - dead branches

Expert Developer Diane

- Surface semantic information
  - complete call-graph
  - dead branches
  - (non-)nullness of arguments

Expert Developer Diane

- Surface semantic information
  - complete call-graph
  - dead branches
  - (non-)nullness of arguments
  - ...

Expert Developer Diane

- Surface semantic information
  - complete call-graph
  - dead branches
  - (non-)nullness of arguments
  - ...

- Static analysis as treasure trove of information usually hidden from Diane

Expert Developer Diane

- To drill-down on warnings locally enable

Expert Developer Diane

- To drill-down on warnings locally enable
  - (more) context-sensitivity

Expert Developer Diane

- To drill-down on warnings locally enable
  - (more) context-sensitivity
  - (more) path-sensitivity

Expert Developer Diane

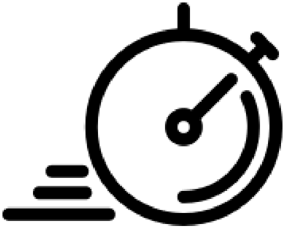
- To drill-down on warnings locally enable
  - (more) context-sensitivity
  - (more) path-sensitivity
  - (more) concurrency-sensitivity

Expert Developer Diane

- To drill-down on warnings locally enable
  - (more) context-sensitivity
  - (more) path-sensitivity
  - (more) concurrency-sensitivity
  - (more) expressive abstract domains

... and beyond soundness:

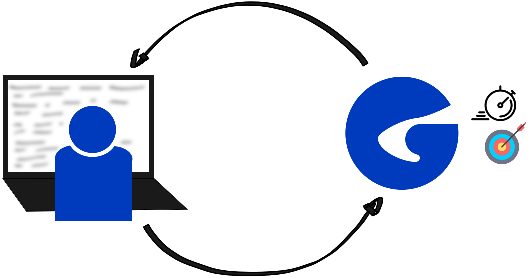
- Ask Diane why some alarm is false (e.g., "these two pointers can never alias")
- Refine analysis relying on this information.



...while limiting recomputation!

Thank you!

- Precise incremental abstract interpretation of multi-threaded programs
- Ideas and ingredients for interactive abstract interpretation



Supported in part by Deutsche Forschungsgemeinschaft DFG (378803395&503812980); the Estonian Research Council (PSG61); the Shota Rustaveli National Science Foundation of Georgia (FR-21-7973); the Estonian Centre of Excellence in IT (EXCITE), funded by the European Regional Development Fund; and the European Union and the Estonian Research Council (TEM-TA119). This research is also supported in part by the National Research Foundation, Singapore, and Cyber Security Agency of Singapore under its National Cybersecurity R&D Programme (Fuzz Testing NRF-NCR25-Fuzz-0001). Any opinions, findings and conclusions, or recommendations expressed in this material are those of the author(s) and do not reflect the views of National Research Foundation, Singapore, and Cyber Security Agency of Singapore.

Co-funded by the European Union

Investing in your future

Funded by DFG Deutsche Forschungsgemeinschaft German Research Foundation

SHOTA RUSTAVELI NATIONAL SCIENCE FOUNDATION

CSA SINGAPORE

NATIONAL RESEARCH FOUNDATION PRIME MINISTER'S OFFICE SINGAPORE Research . Innovation . Enterprise

European Union European Regional Development Fund

Investing in your future