Static analysis with LiSA

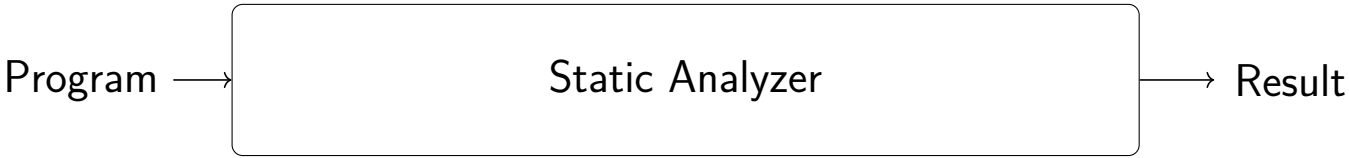
Giacomo Zanatta

giacomo.zanatta@unive.it

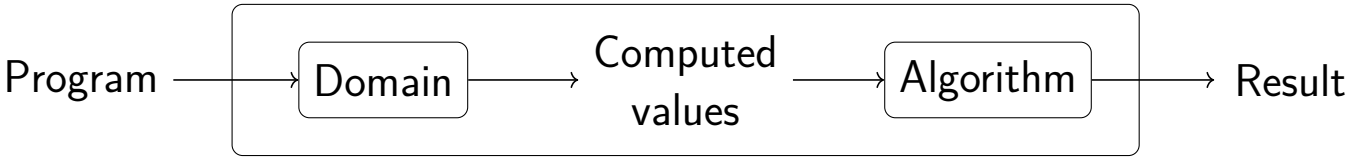
Ca' Foscari University of Venice
Department of Environmental Sciences, Informatics and Statistics

2025

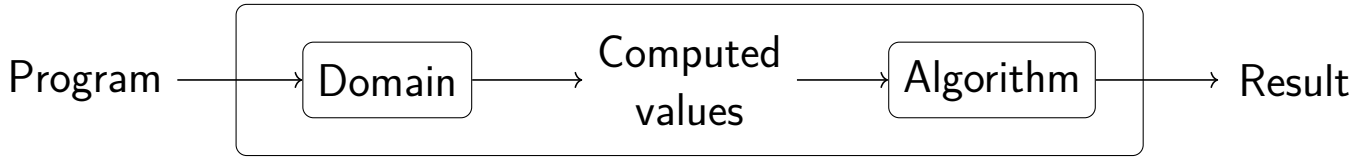
From theory to practice



From theory to practice

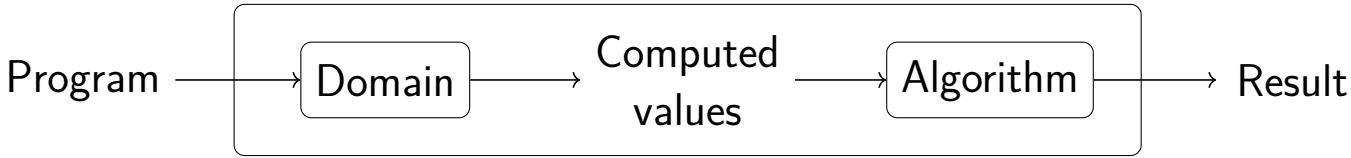


From theory to practice



$$y = (2 * 2) - 4$$
$$z = 1 / y$$

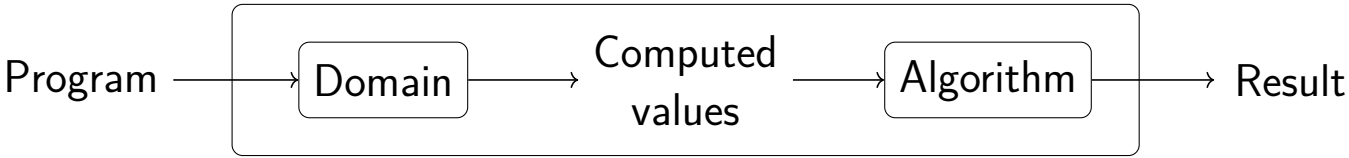
From theory to practice



$y = (2 * 2) - 4$ constant $y \mapsto 0$

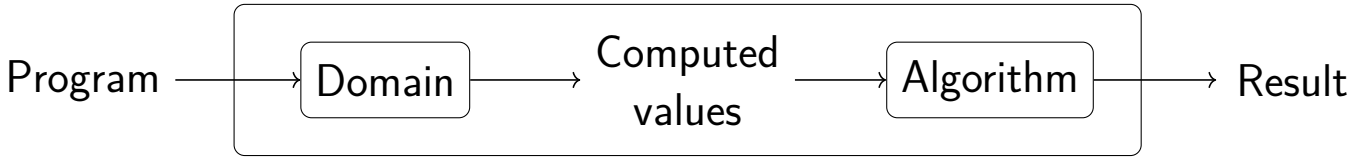
$z = 1 / y$ propagation $z \mapsto$ error

From theory to practice



$y = (2 * 2) - 4$ $z = 1 / y$	constant propagation	$y \mapsto 0$ $z \mapsto$ error	check for div by zero	message to the user

From theory to practice



y = (2 * 2) - 4 z = 1 / y	constant propagation	y ↦ 0 z ↦ error	check for div by zero	message to the user

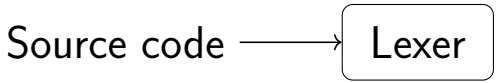
how do we parse the input code?

A (simplified) compiler/interpreter

Source code

$x = y + 2$

A (simplified) compiler/interpreter



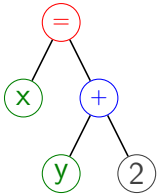
x = y + 2

ID EQ ID
PLUS LITERAL

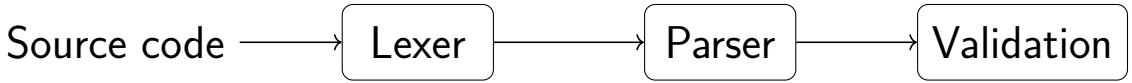
A (simplified) compiler/interpreter

x = y + 2

ID EQ ID
PLUS LITERAL

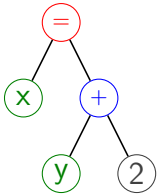


A (simplified) compiler/interpreter



x = y + 2

ID EQ ID
PLUS LITERAL

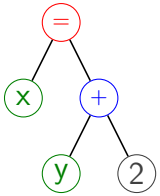


type check,
duplicate functions,
...

A (simplified) compiler/interpreter

x = y + 2

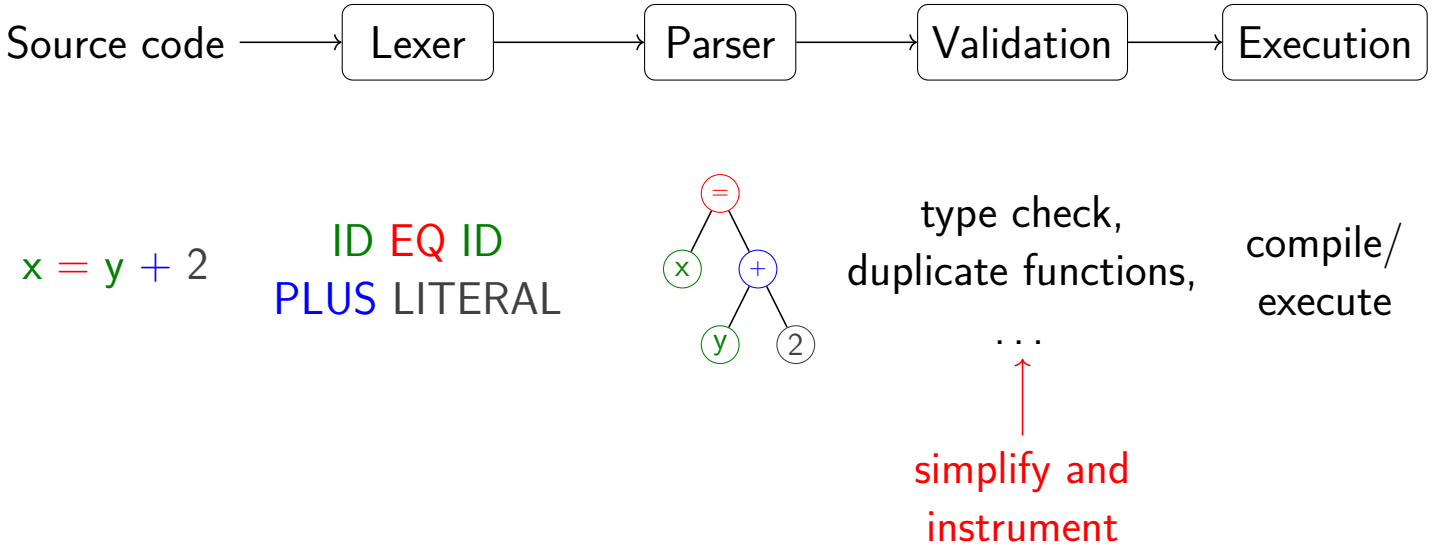
ID EQ ID
PLUS LITERAL



type check,
duplicate functions,
...

compile/
execute

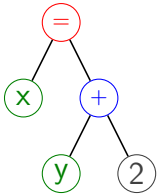
A (simplified) compiler/interpreter



A (simplified) compiler/interpreter

$x = y + 2$

ID EQ ID
PLUS LITERAL



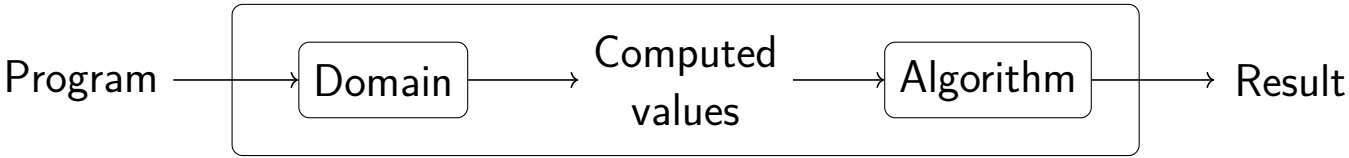
type check,
duplicate functions,
...

simplify and
instrument

compile/
execute

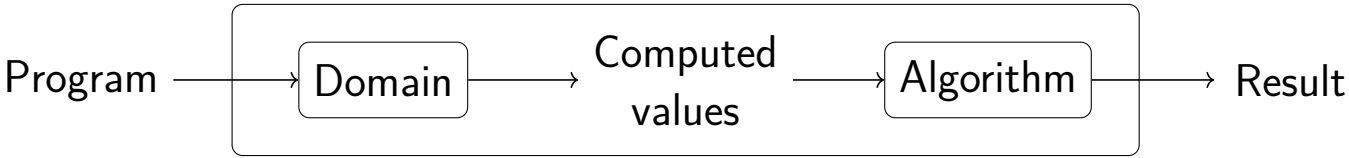
run domains
and algorithms

From theory to practice



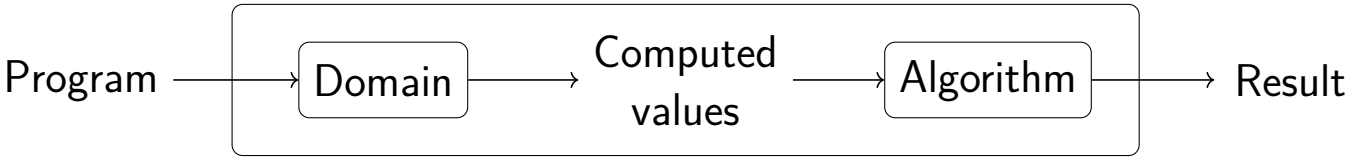
y = (2 * 2) - 4
z = 1 / f(y)

From theory to practice



y = (2 * 2) - 4	constant	$y \mapsto 0$
z = 1 / f(y)	propagation	$z \mapsto ?$

From theory to practice

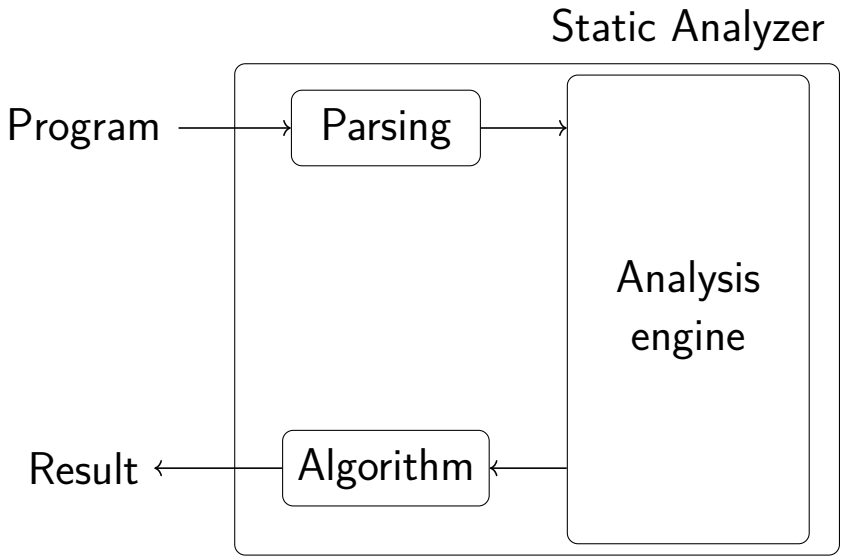


$y = (2 * 2) - 4$ constant $y \mapsto 0$

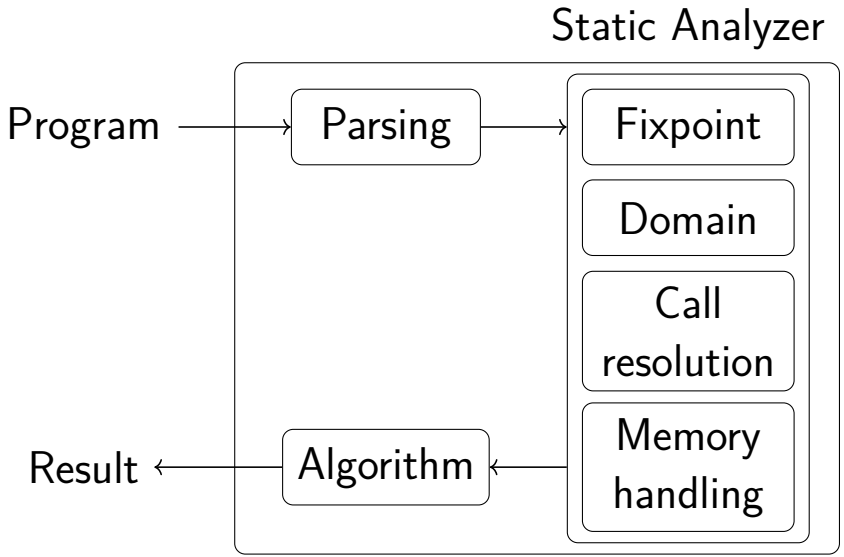
$z = 1 / f(y)$ propagation $z \mapsto ?$

need to handle some program features independently of the domain

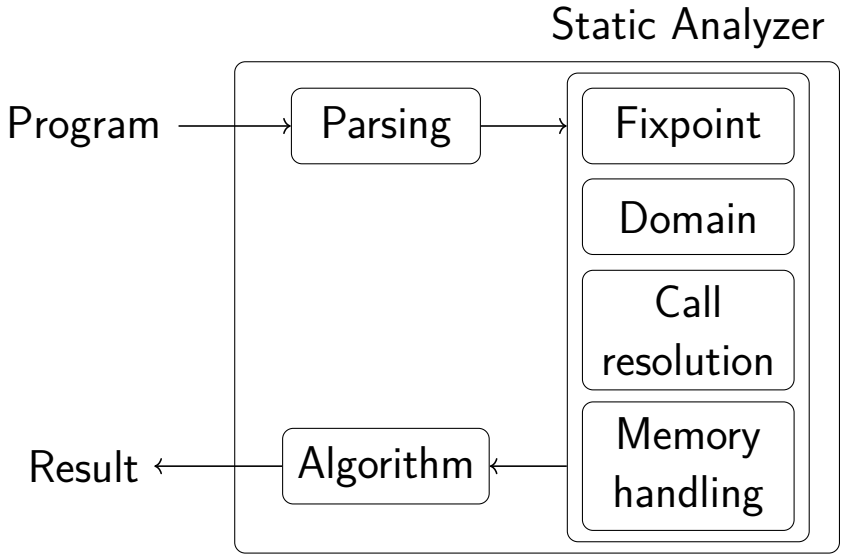
A realistic structure



A realistic structure



A realistic structure



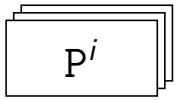
Key points:

- separate components for each task
- modular components

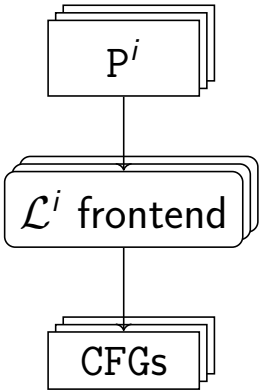
LiSA, a Library for Static Analysis

- Open source library written in Java
- Created and maintained by the SSV group @ UniVE
- Library for creating static analyzers based on abstract interpretation
  - For multiple programming languages
  - For a variety of different domains

LiSA Overview



LiSA Overview



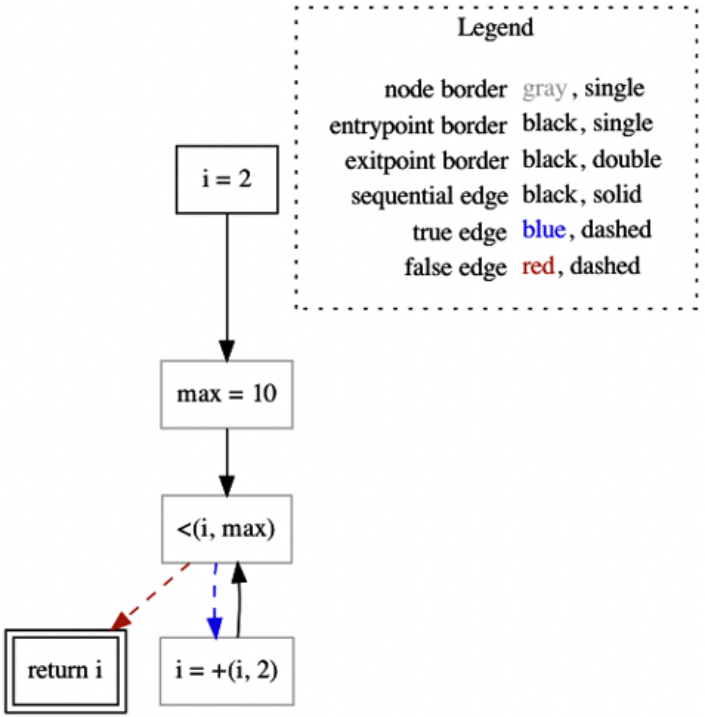
CFGs

A `CFG` is:

- Set of nodes (`Statement`)
- Set of edges linking nodes
  - `SequentialEdge`
  - `TrueEdge` (the statement at source node is assumed to hold)
  - `FalseEdge` (the statement at source node is assumed not to hold)

Example:

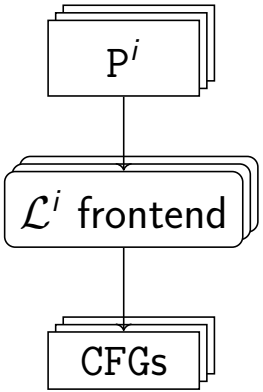
```
1 int i = 2;
2 int max = 10;
3 while (i < max)
4   i = i + 2;
5 return i;
```



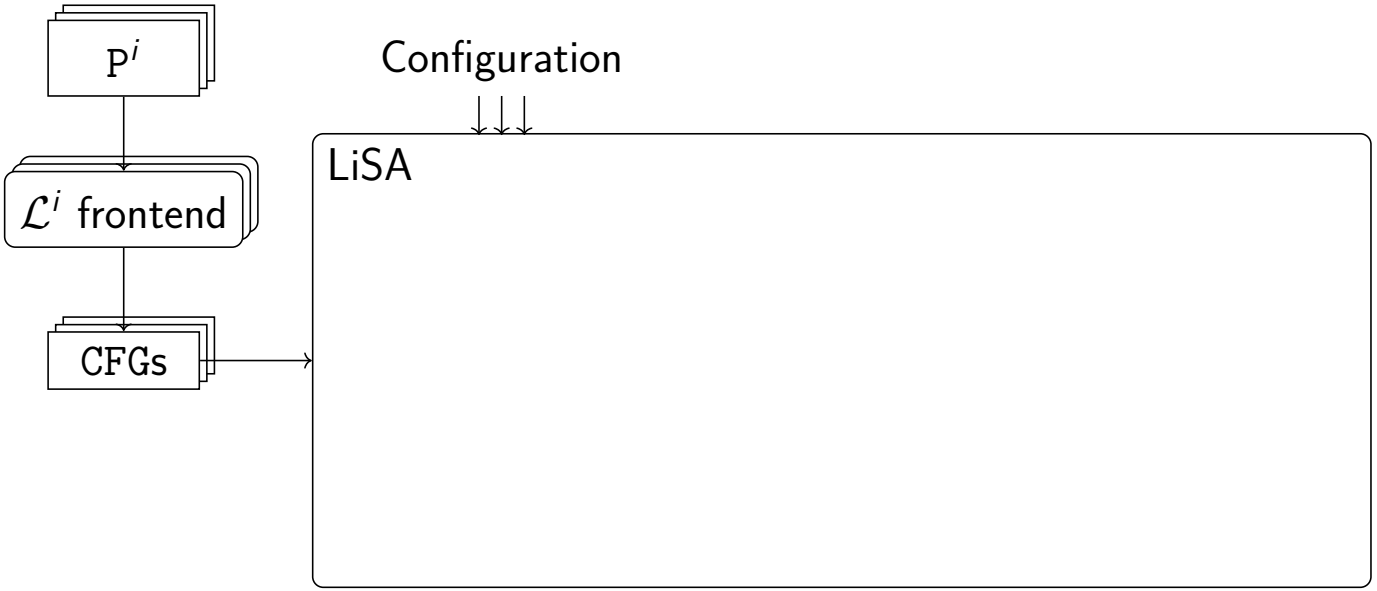
Why CFGs?

- Get rid of the syntax for control flow
- Extensible: node instances are not fixed
- Received as inputs: no knowledge of source languages is kept past the frontends
- Easy to define fixpoints over graphs!
- Can be produced in isolation: each frontend produces the `CFG`s for that language, and we can analyze them together
    - For the course, we will use a frontend for an IMP language (similar to JAVA)

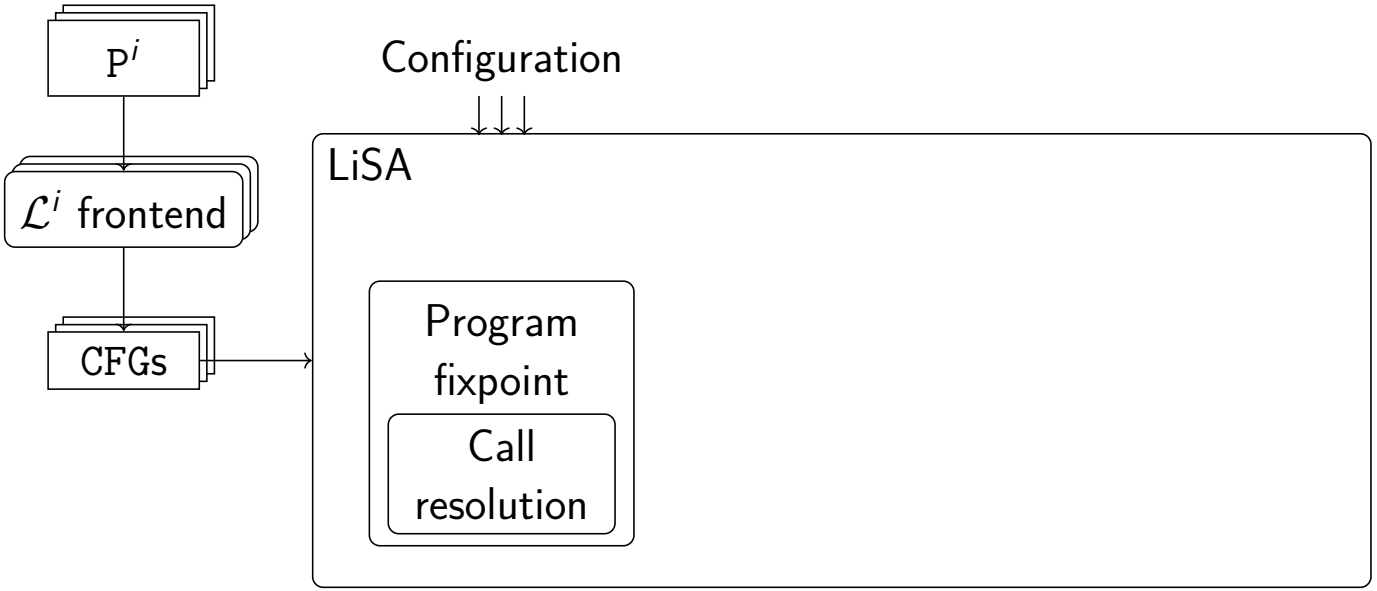
LiSA Overview



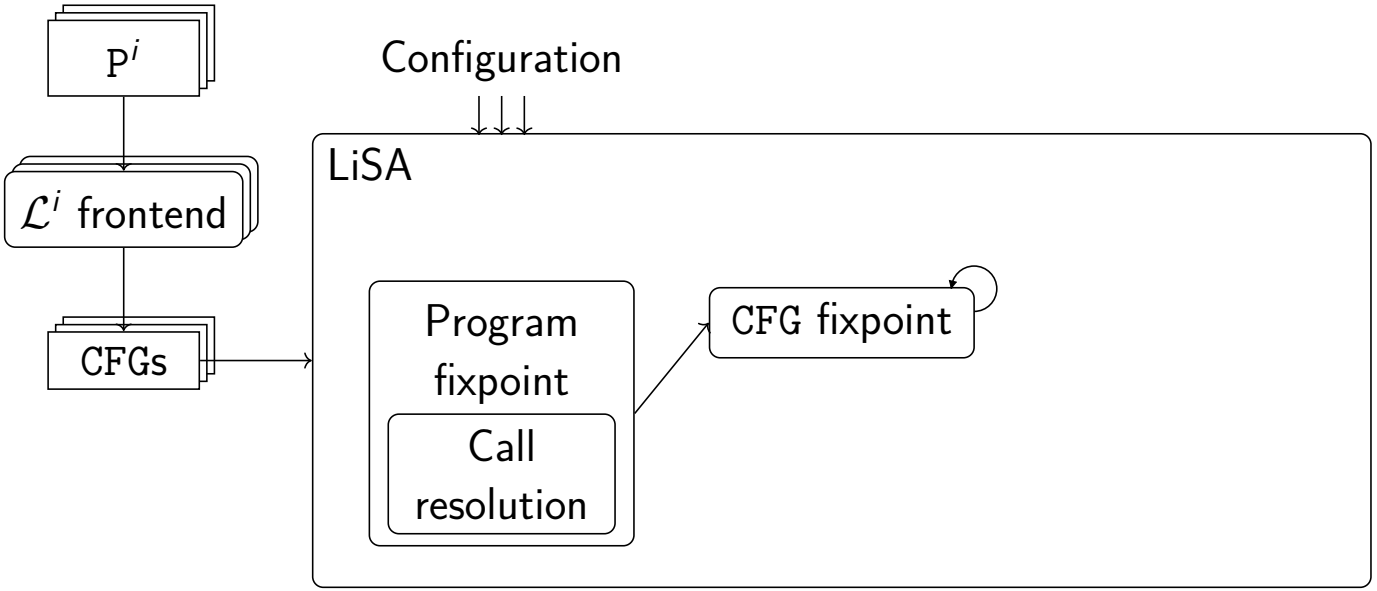
LiSA Overview



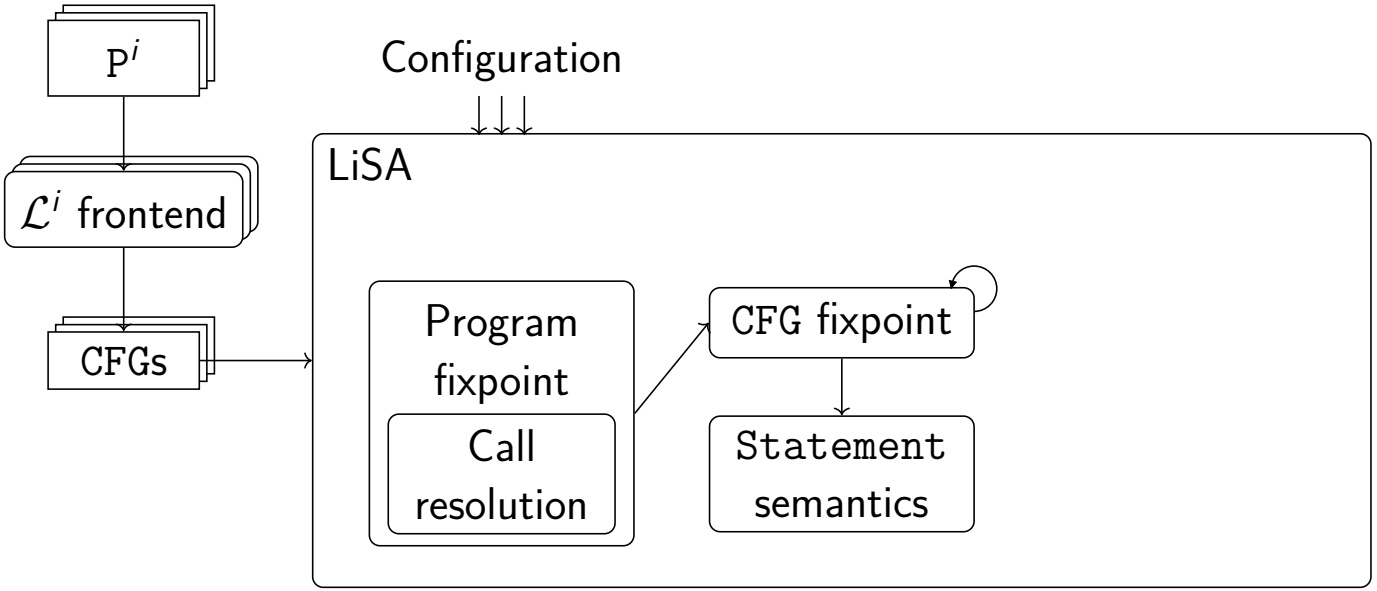
LiSA Overview



LiSA Overview



LiSA Overview



Syntax vs semantics

- `Statement` (`CFG` nodes) express the syntax of the program
- A statement can be complex: `x == null ? "unknown": x.toString()`
- Different statements can have the same meaning (semantics): `"a" + "b"` and `"a".concat("b")`
- Can we simplify?

Syntax vs semantics

- `Statement` (`CFG` nodes) express the syntax of the program
- A statement can be complex: `x == null ? "unknown": x.toString()`
- Different statements can have the same meaning (semantics): `"a" + "b"` and `"a".concat("b")`
- Can we simplify? Yes! The `SymbolicExpressions` language:
  - Each symbolic expression is an "atomic" operation (concatenation, addition, ...)
  - Domains handle `SymbolicExpressions` instead of `Statements`
  - Fixpoint uses `Statement.semantics()` to rewrite each `Statement` to `SymbolicExpression`(s) and feed them to the domain

Examples (pseudocode)

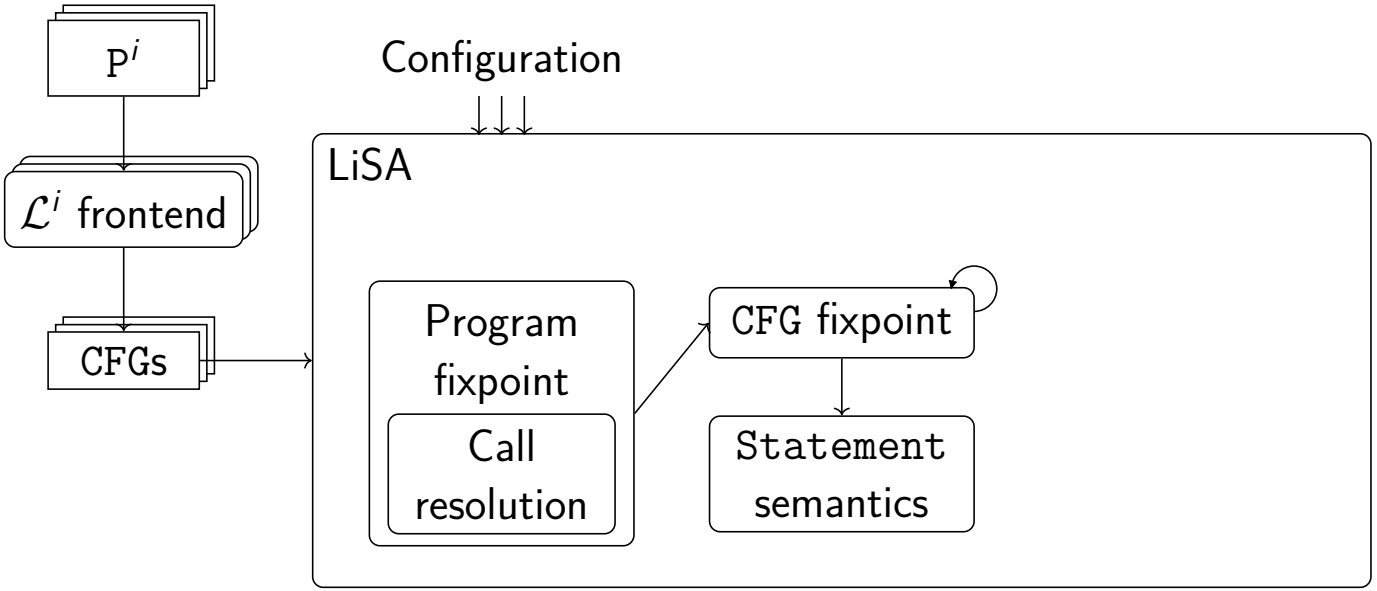
```
Addition.semantics():
  return domain.eval(new ArithmSum(left, right))
```

```
Concatenation.semantics():
  return domain.eval(new StringConcat(left, right))
```

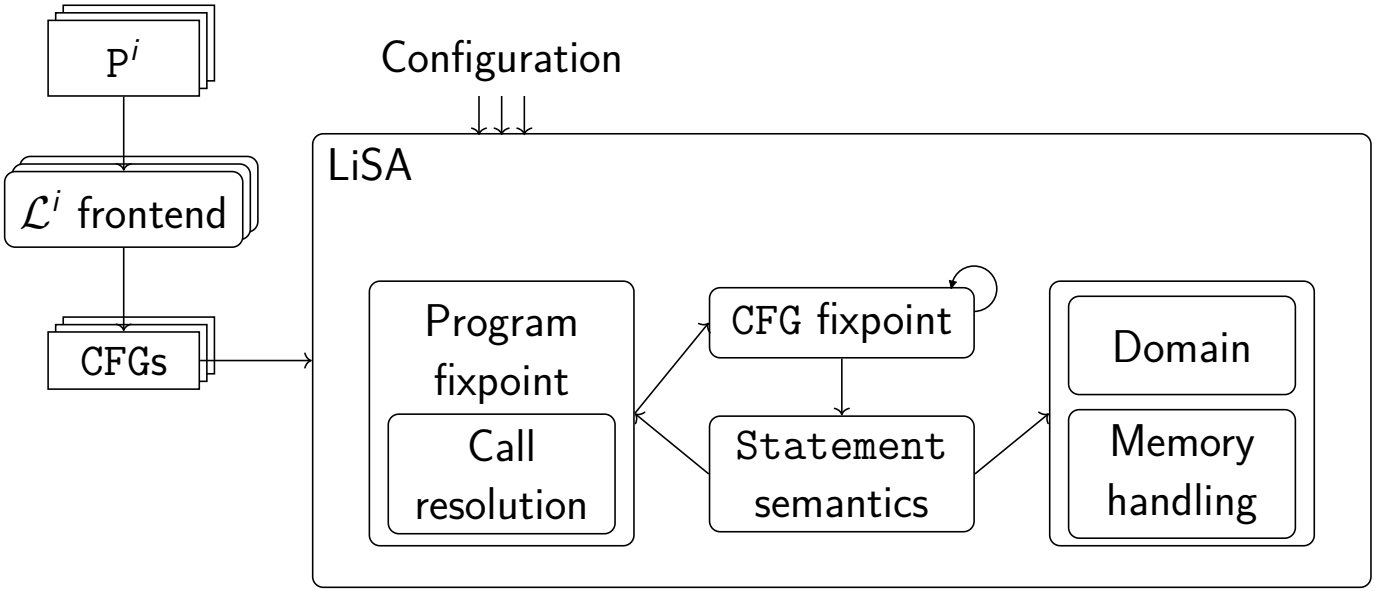
```
Plus.semantics():
  if left is string or right is string:
    return domain.eval(new StringConcat(left, right))
  else:
    return domain.eval(new ArithmSum(left, right))
```

```
Conditional.semantics():
  if domain.isSatisfied(condition):
    return ifTrue.semantics()
  else:
    return ifFalse.semantics()
```

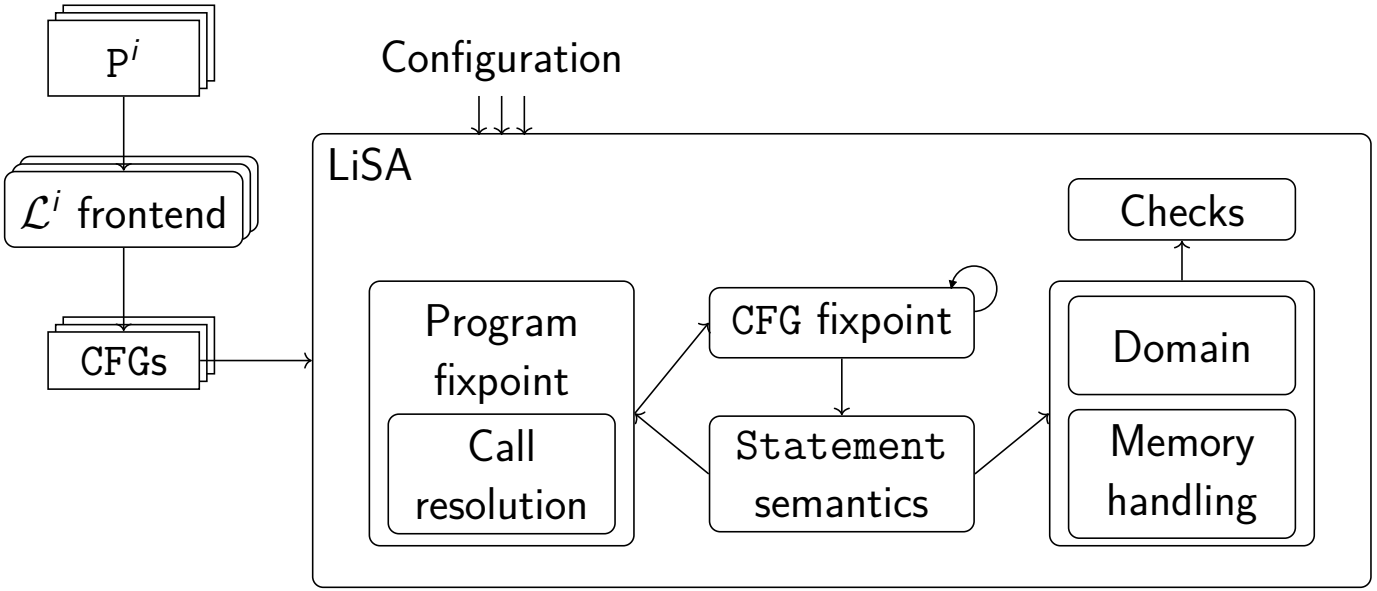
LiSA Overview



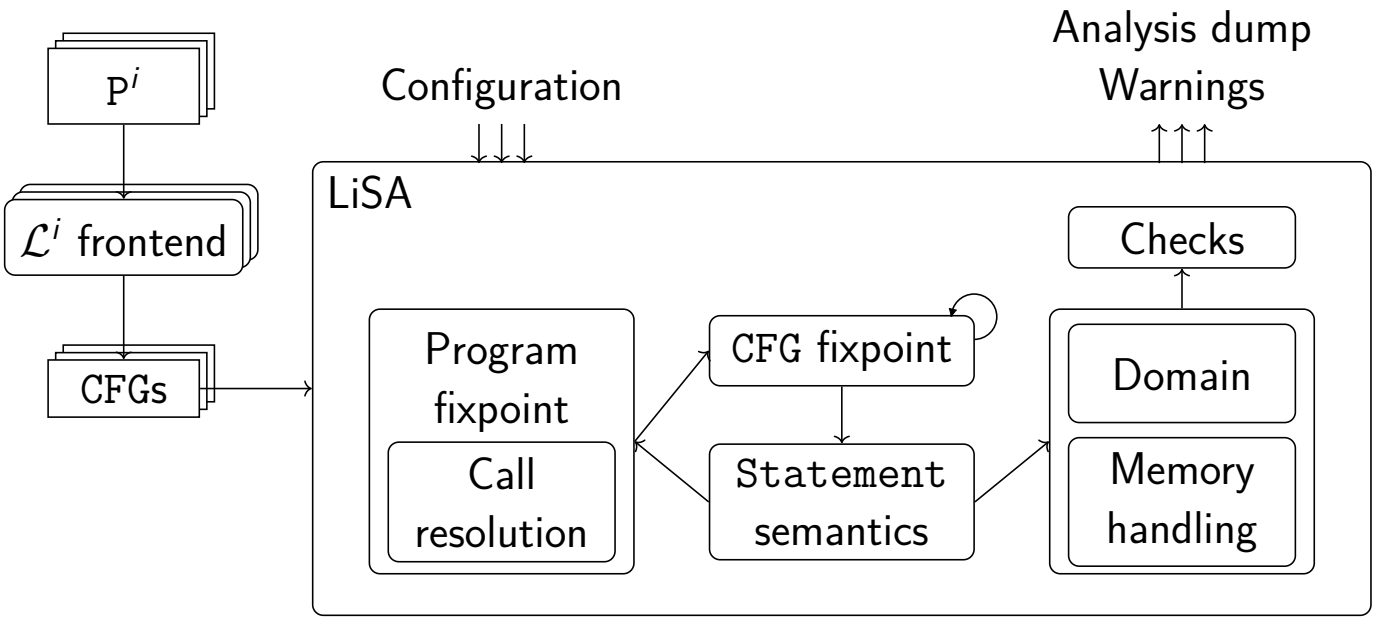
LiSA Overview



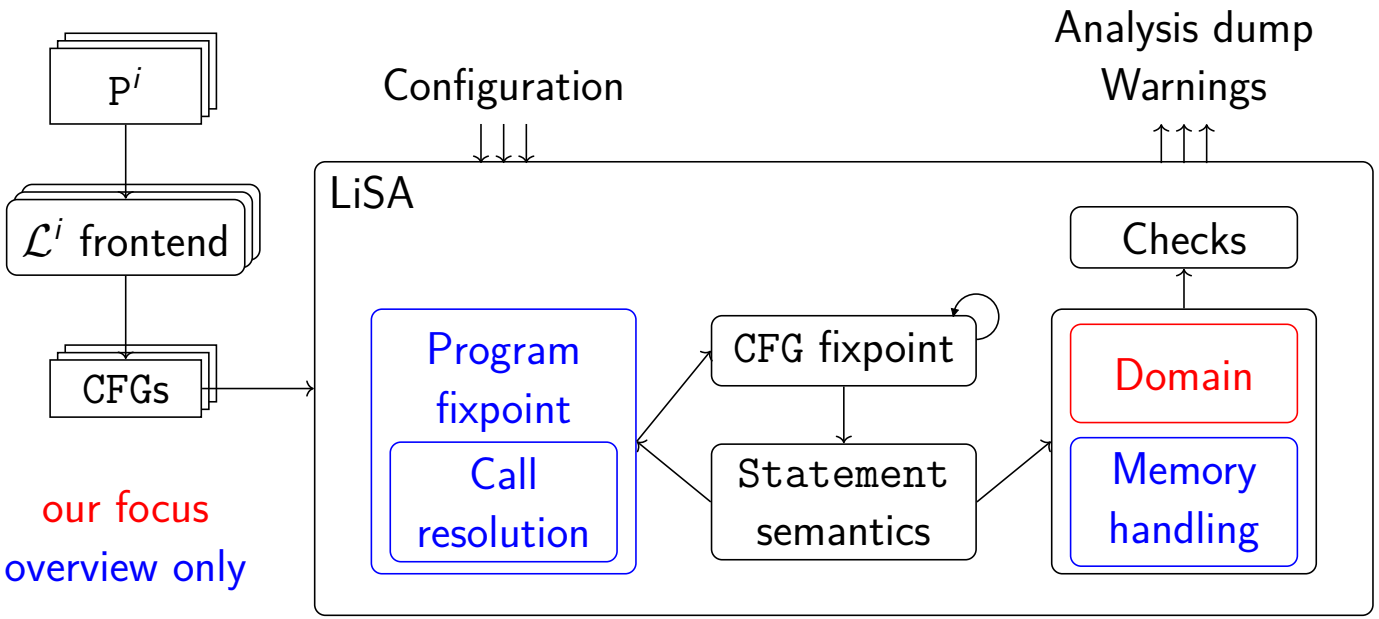
LiSA Overview



LiSA Overview



LiSA Overview



Questions?

Course project setup

Coding demo

This week's task

Download the course project and ensure that you can get it up and running

- Send me an email with your **GitHub username** to get permissions to the repo
- Create a branch named `id-lastname-firstname` and **checkout the branch**
  - e.g., `859156-zanatta-giacomo`
- Run `gradlew build` to ensure everything is working properly
- Import the project in your IDE
- Run the test class to ensure everything works properly

Links and Tips

- Repository for the tasks
- Cloning the repository:
  `git clone git@github.com:giacomozanatta/scsr-2025.git`
- Create a branch: `git checkout -b <branch name>`
- Commit your code: `git commit -am "<commit message>"`
- Push your code: `git push origin <branch name>`

Enjoy coding!

giacomo.zanatta@unive.it

https://github.com/lisa-analyzer/lisa

https://github.com/giacomozanatta/scsr-2025