# Incrementalizing Static Program Analyses

Tamás Szabó, Gábor Bergmann, Sebastian Erdweg, Markus Völter

itemis

TU Delft

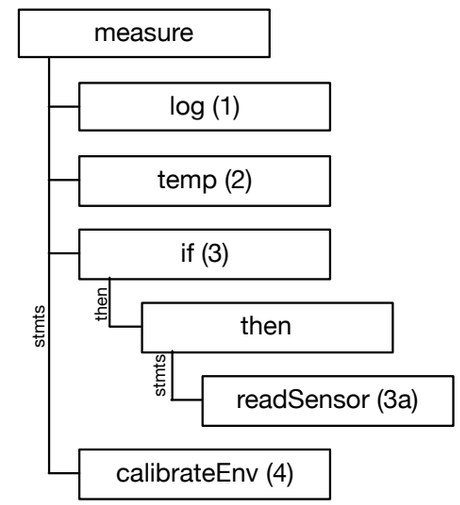
MŰEGYETEM 1782

# Analyses are essential in IDEs

## Subject Program

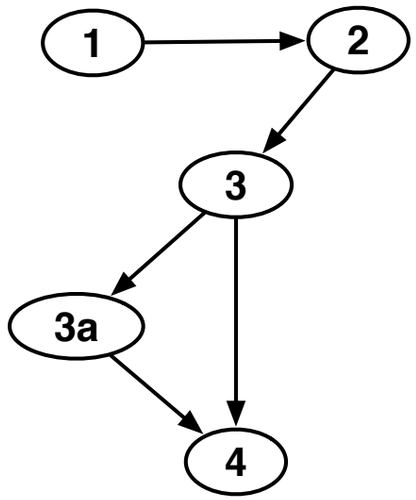
```
void measure(Environment env) {
  log("Starting measurement"); (1)
  int32 temp;                  (2)
  if (env.active) {            (3)
    readSensor(&temp);         (3a)
  } if
  calibrateEnv(env, temp);     (4)
} measure (function)
```

## AST




Interactive editing requires *incremental* maintenance

## CFG / DFG




## Program Analyses

- Uninitialized read
- Liveness
- Sign
- …

# Anatomy of CFlow

```
public void test() {
  Object var1;
  Object var2 = new Object();
  if (Math.random() > 0.5) {
    var1 = null;
  } else {
    var1 = new Object();
  }
  System.out.println(var1.toString());
  System.out.println(var2.toString());
}
```

# Anatomy of CFlow

```
public void test() {
  Object var1;
  Object var2 = new Object();
  if (Math.random() > 0.5) {
    var1 = null;
  } else {
    var1 = new Object();
  }
  System.out.println(var1.toString());
  System.out.println(var2.toString());
}
```

# Anatomy of CFlow

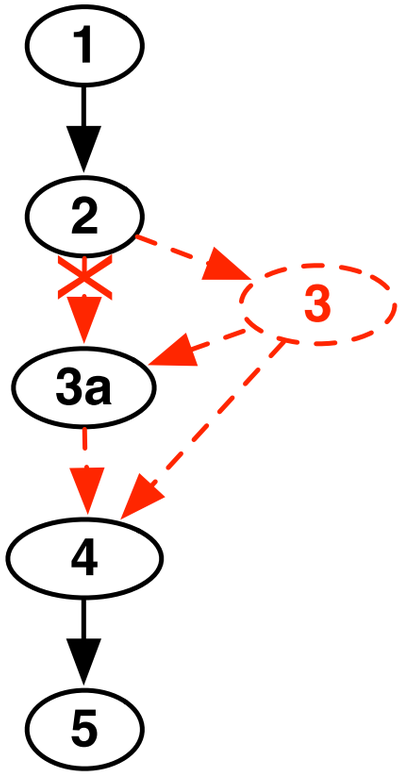
```
public void test() {
  Object var1;
  Object var2 = new Object();
  if (Math.random() > 0.5) {
    var1 = null;
  } else {
    var1 = new Object();
  }
  System.out.println(var1.toString());
  System.out.println(var2.toString());
}
```

# Anatomy of CFlow

```
public void test() {
  Object var1;
  Object var2 = new Object();
  if (Math.random() > 0.5) {
    var1 = null;
  } else {
    var1 = new Object();
  }
  System.out.println(var1.toString());
  System.out.println(var2.toString());
}
```

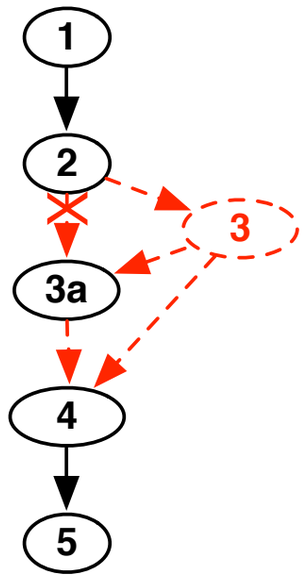
# IncA Evaluation

```
Object var1;                                  (1)
Object var2 = new Object();                   (2)
if (Math.random() > 0.5) {                    (3)
  var1 = null;                                (3a)
}
System.out.println(var1.toString());          (4)
System.out.println(var2.toString());          (5)
```



# IncA Evaluation

```
Object var1;                                  (1)
Object var2 = new Object();                   (2)
if (Math.random() > 0.5) {                    (3)
  var1 = null;                                (3a)
}
System.out.println(var1.toString());          (4)
System.out.println(var2.toString());          (5)
```




+

1, 2, 3a, 3, 4, 5

**Stmt**

**NextStmt**

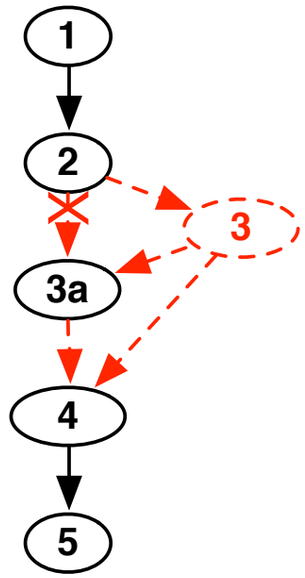
| src | trg | |
|---|---|---|
| 1 | 2 | |
| 2 | 3a | - |
| 3a | 4 | |
| 4 | 5 | |
| 2 | 3 | + |
| 3 | 4 | + |

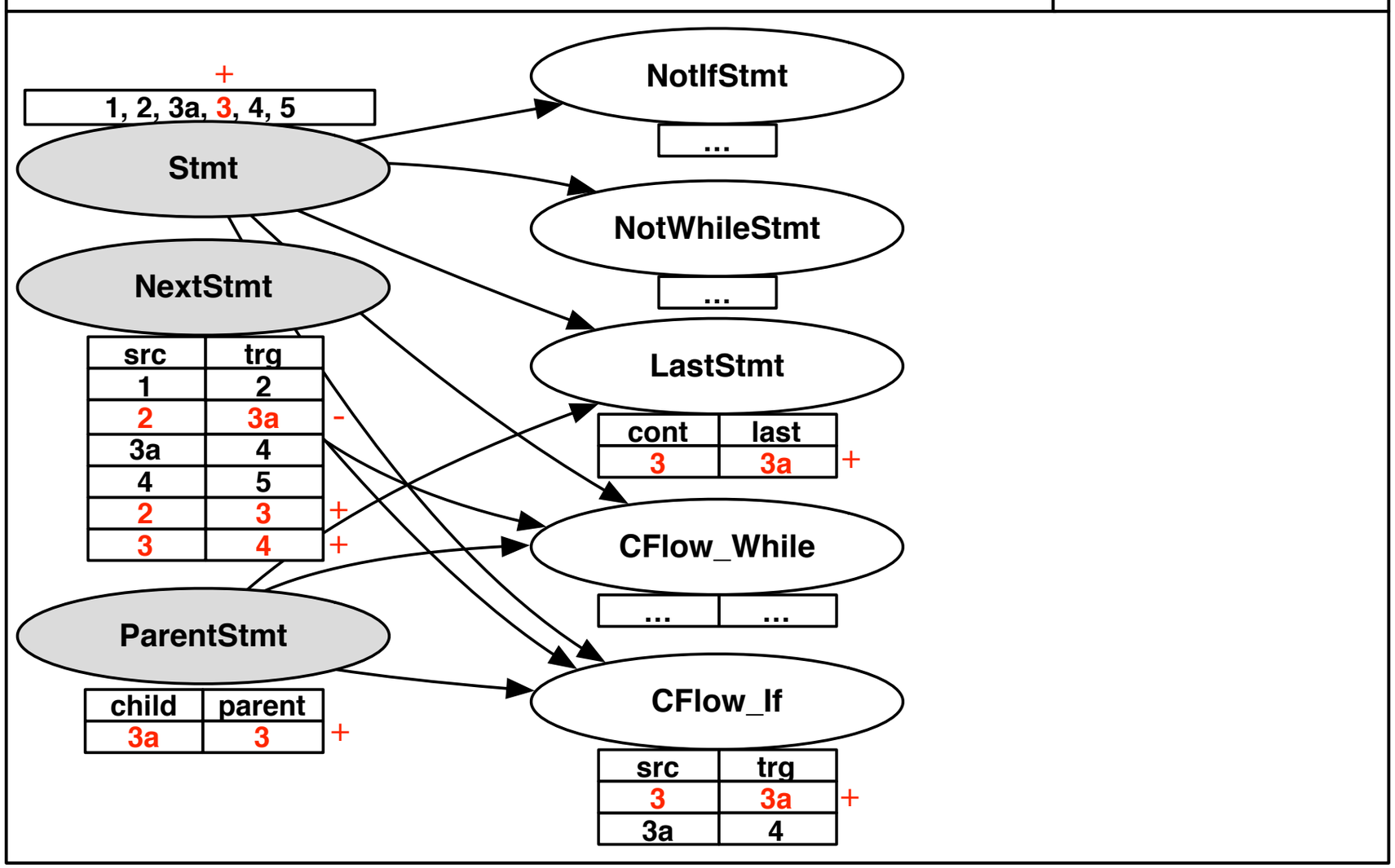
**ParentStmt**

| child | parent | |
|---|---|---|
| 3a | 3 | + |

# IncA Evaluation

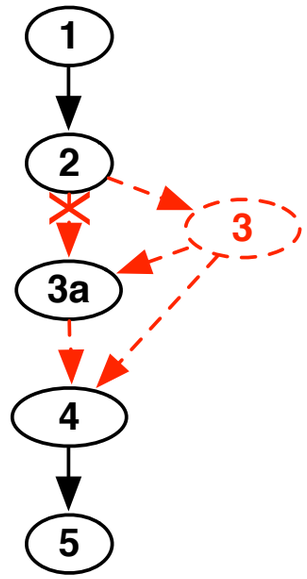
```
Object var1;                                  (1)
Object var2 = new Object();                   (2)
if (Math.random() > 0.5) {                    (3)
  var1 = null;                                (3a)
}
System.out.println(var1.toString());          (4)
System.out.println(var2.toString());          (5)
```

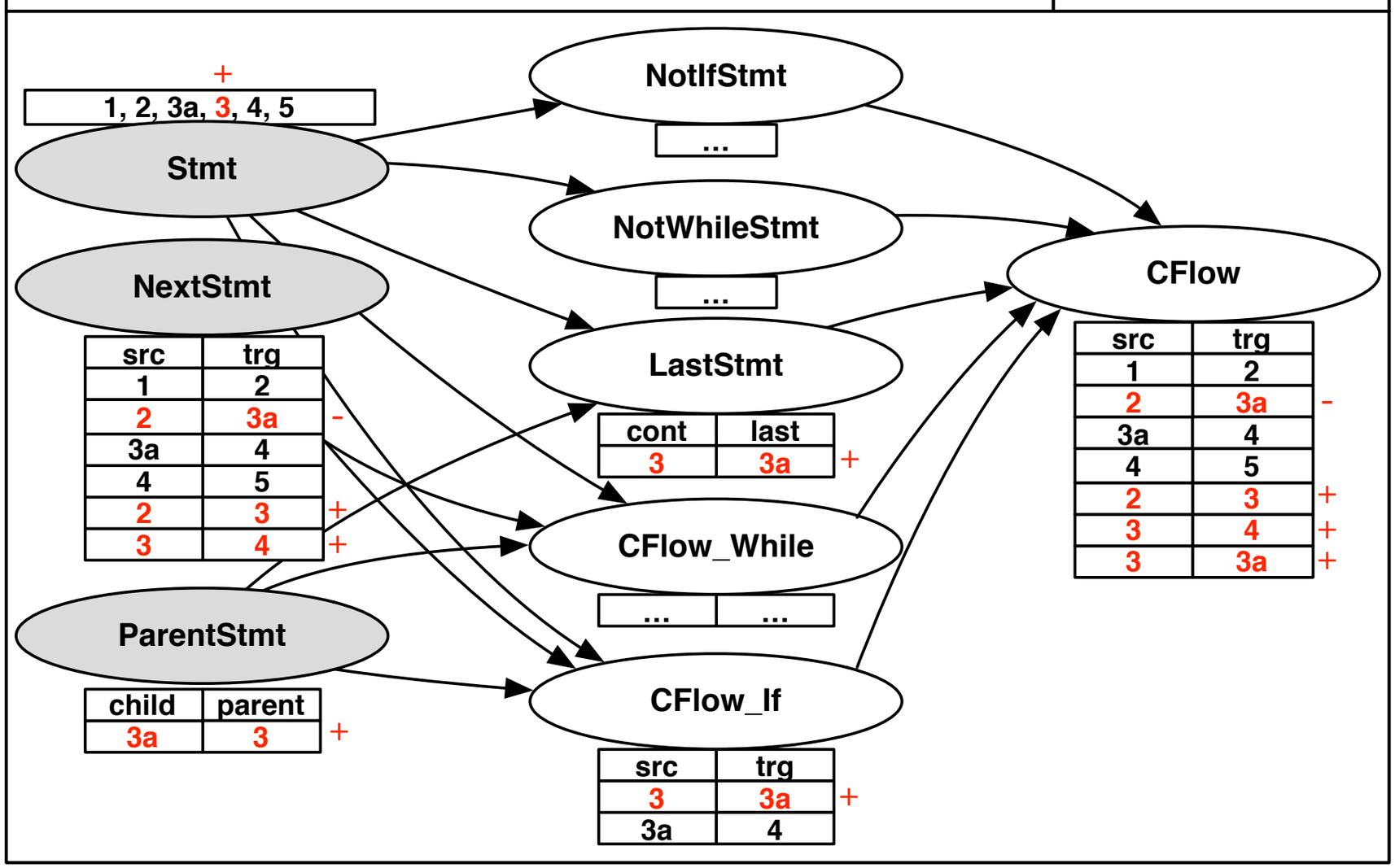


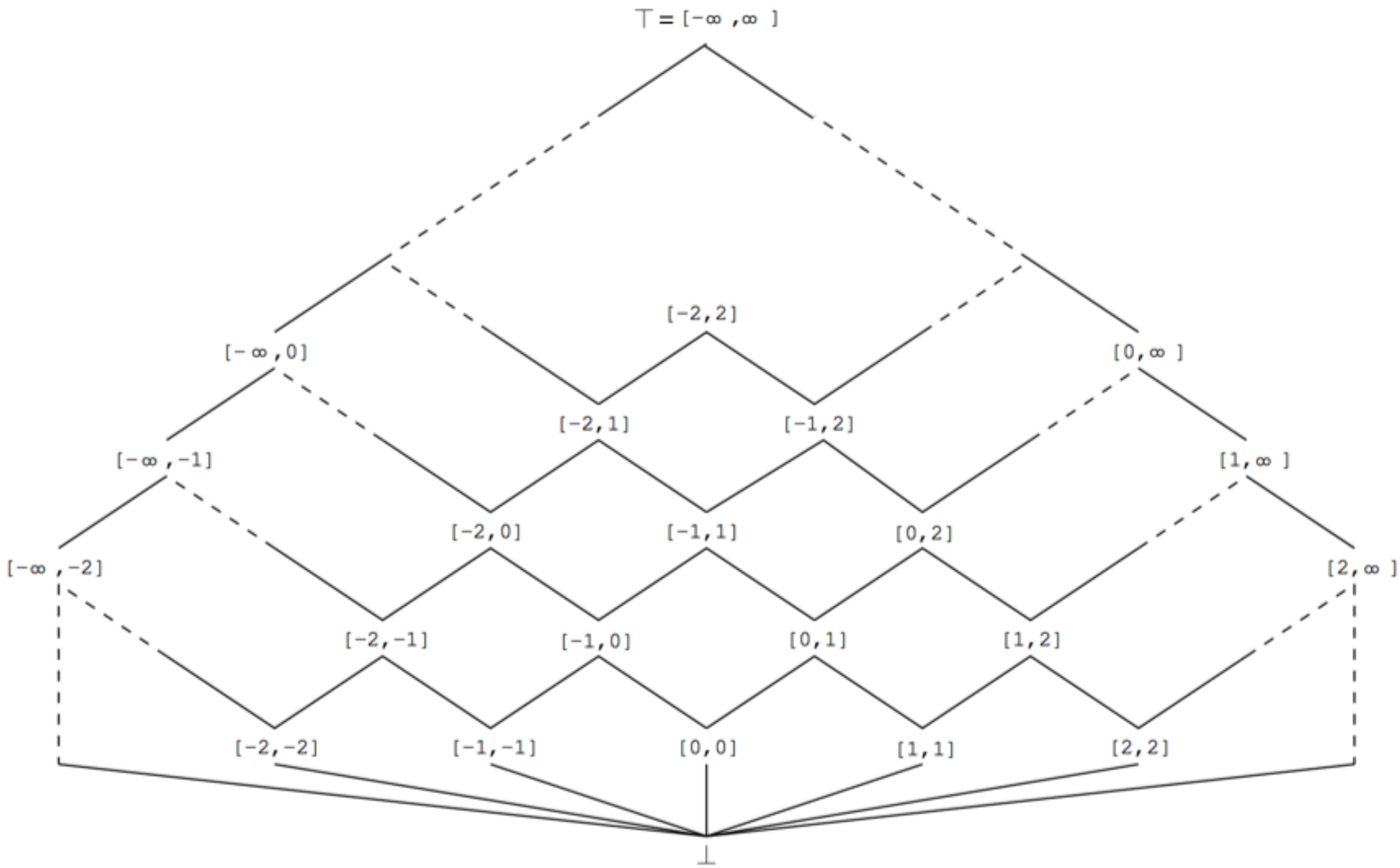
# IncA Evaluation

```
Object var1;                                  (1)
Object var2 = new Object();                   (2)
if (Math.random() > 0.5) {                    (3)
  var1 = null;                                (3a)
}
System.out.println(var1.toString());          (4)
System.out.println(var2.toString());          (5)
```

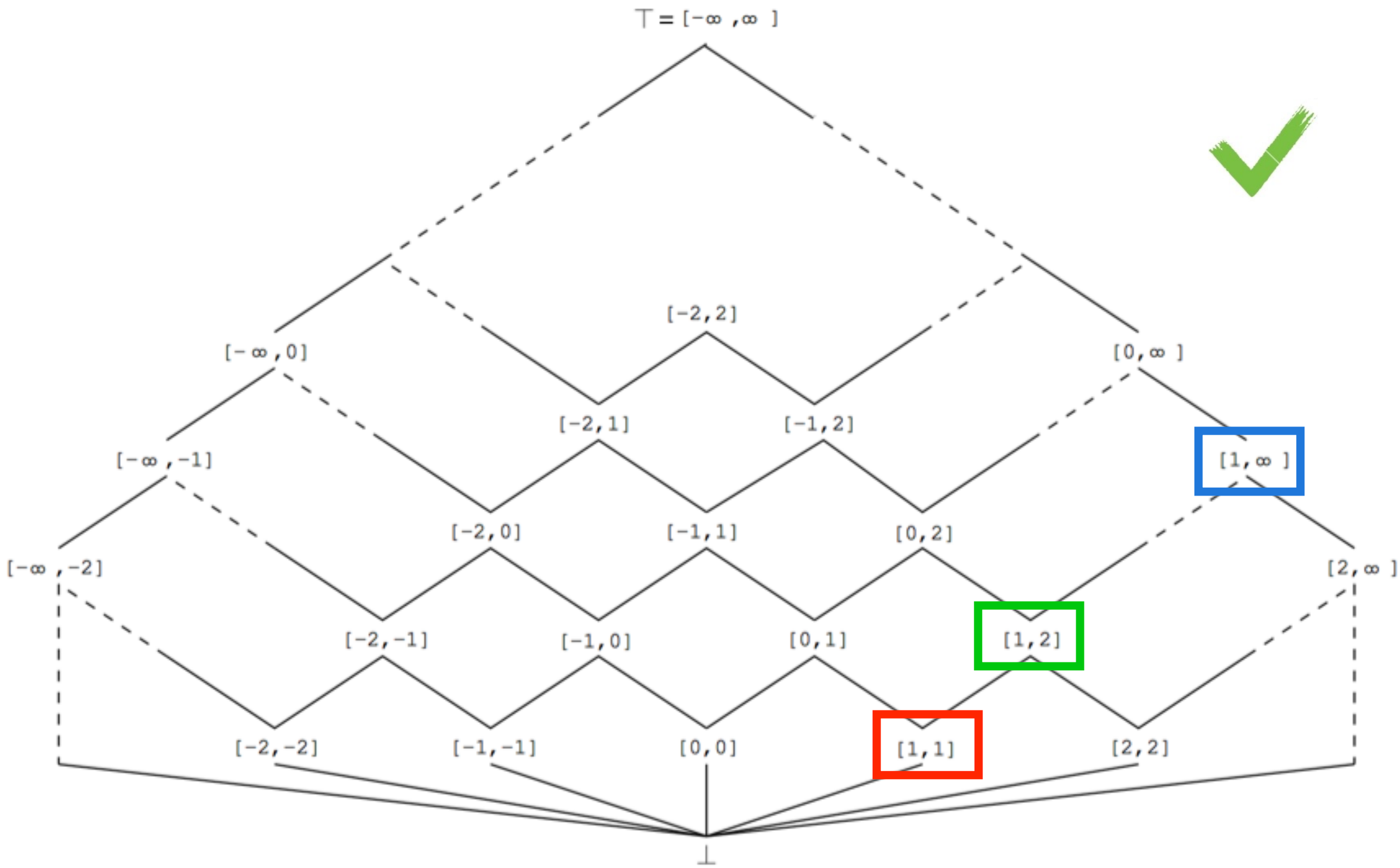


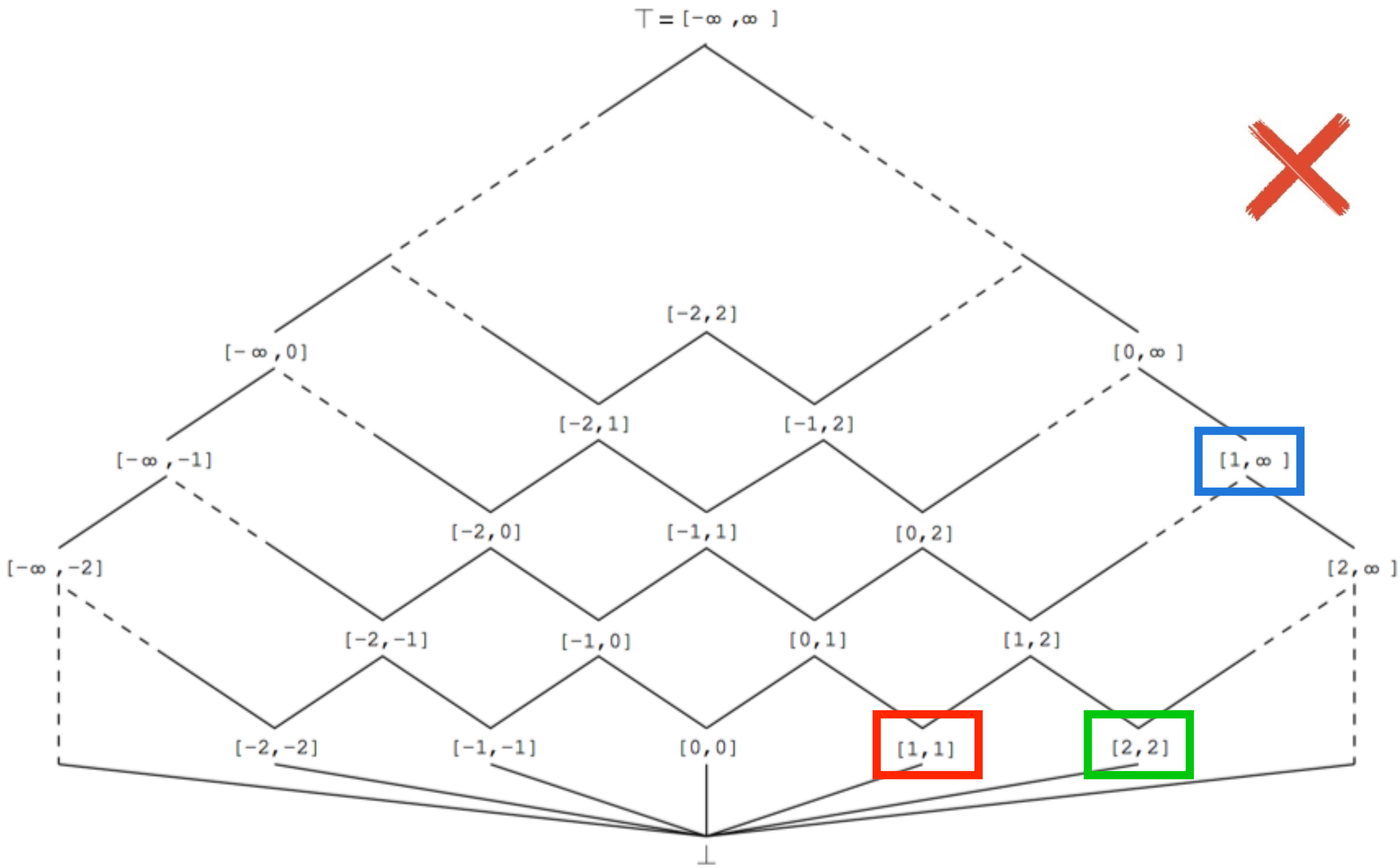
# Interval Lattice



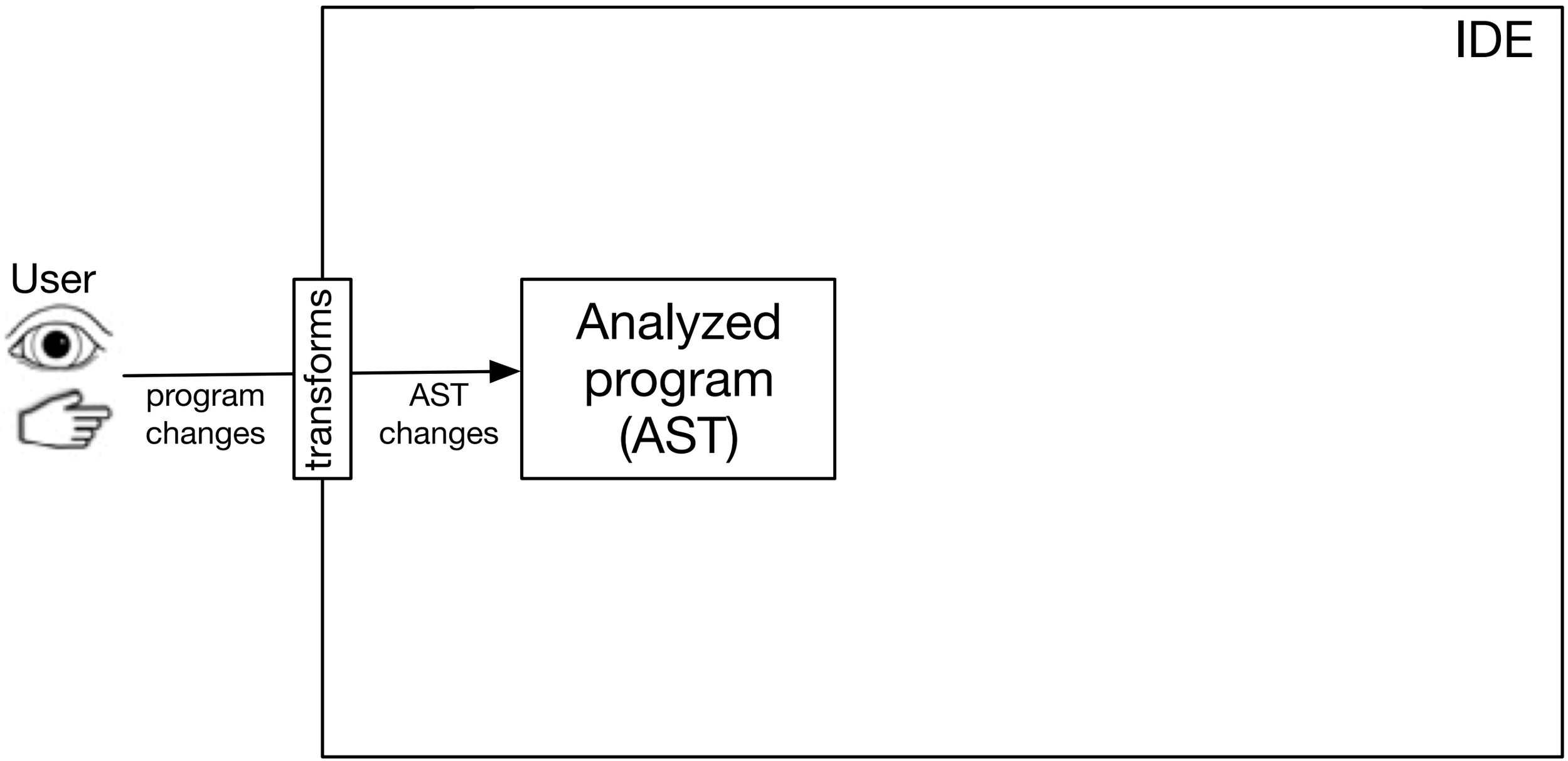
# Interval Lattice



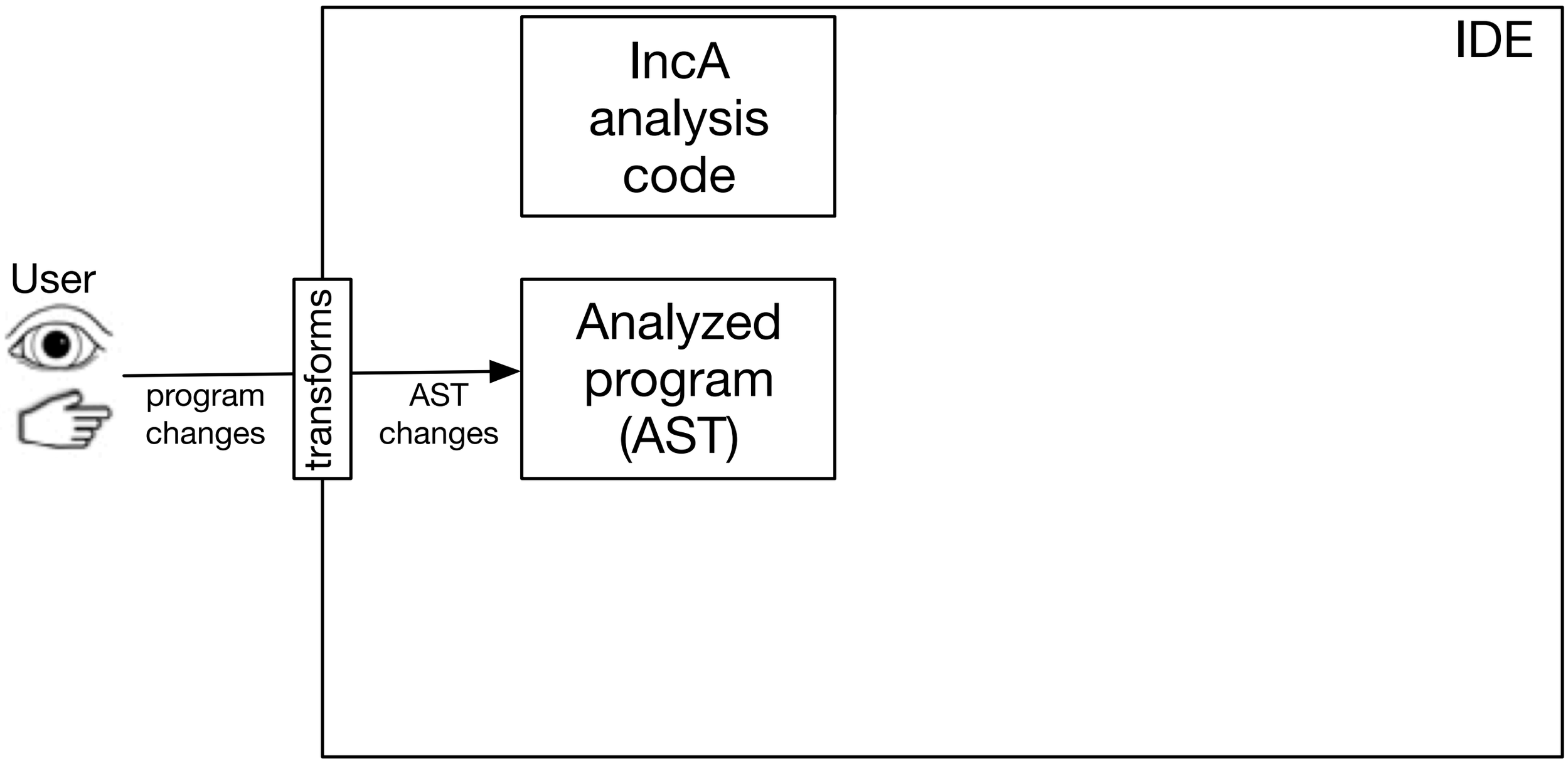
# Interval Lattice



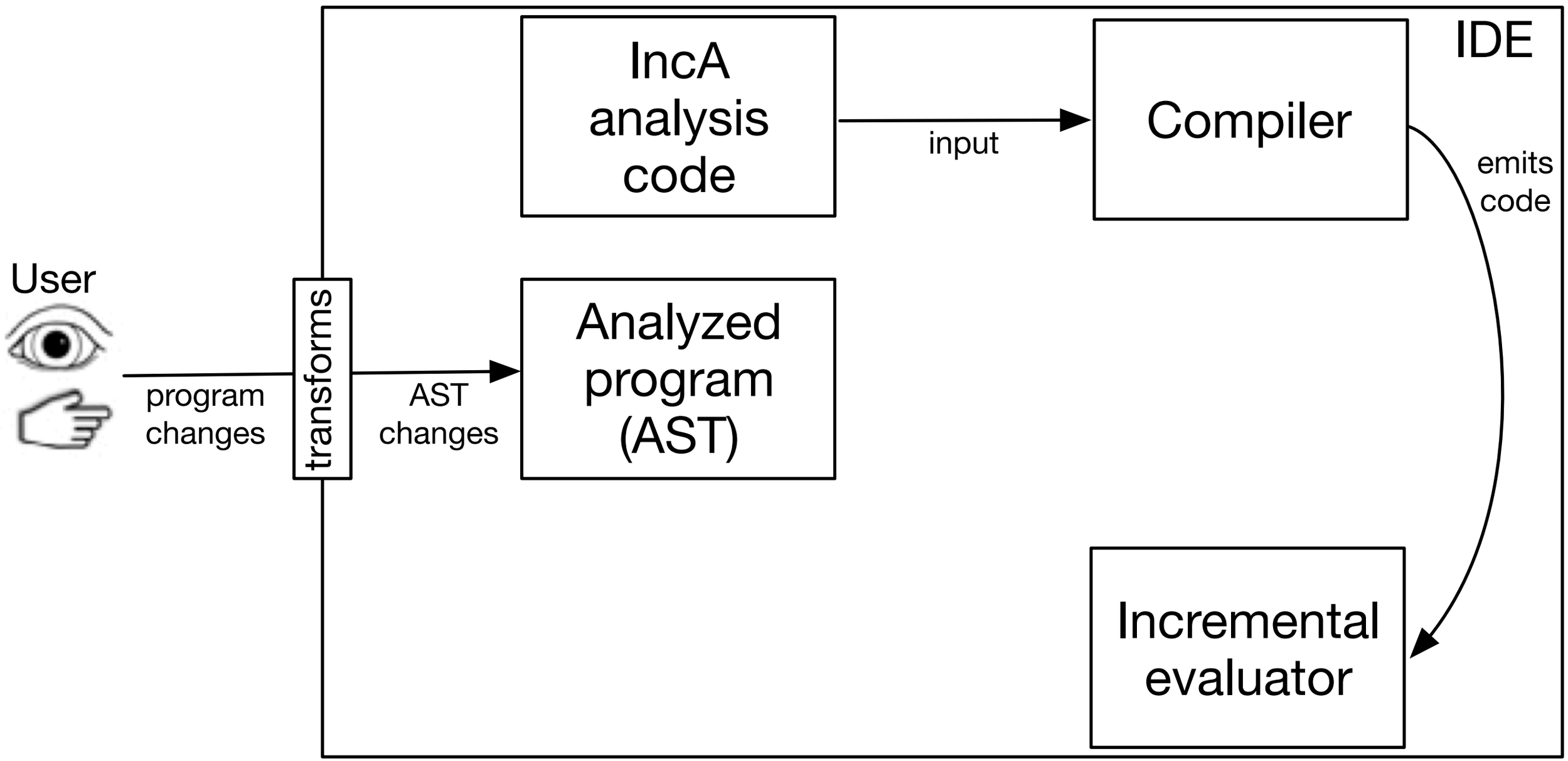
# IncA - Architecture



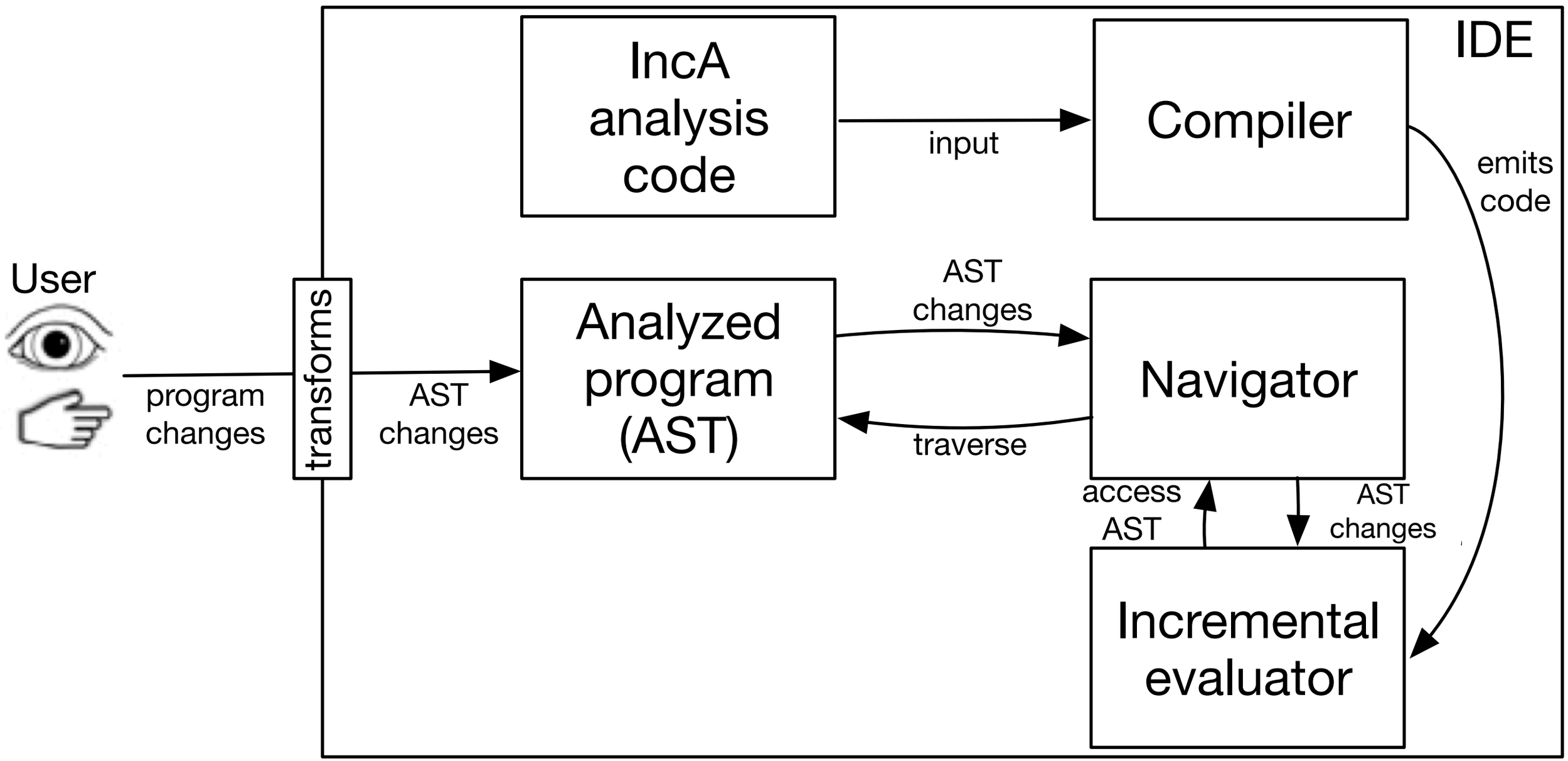
# IncA - Architecture



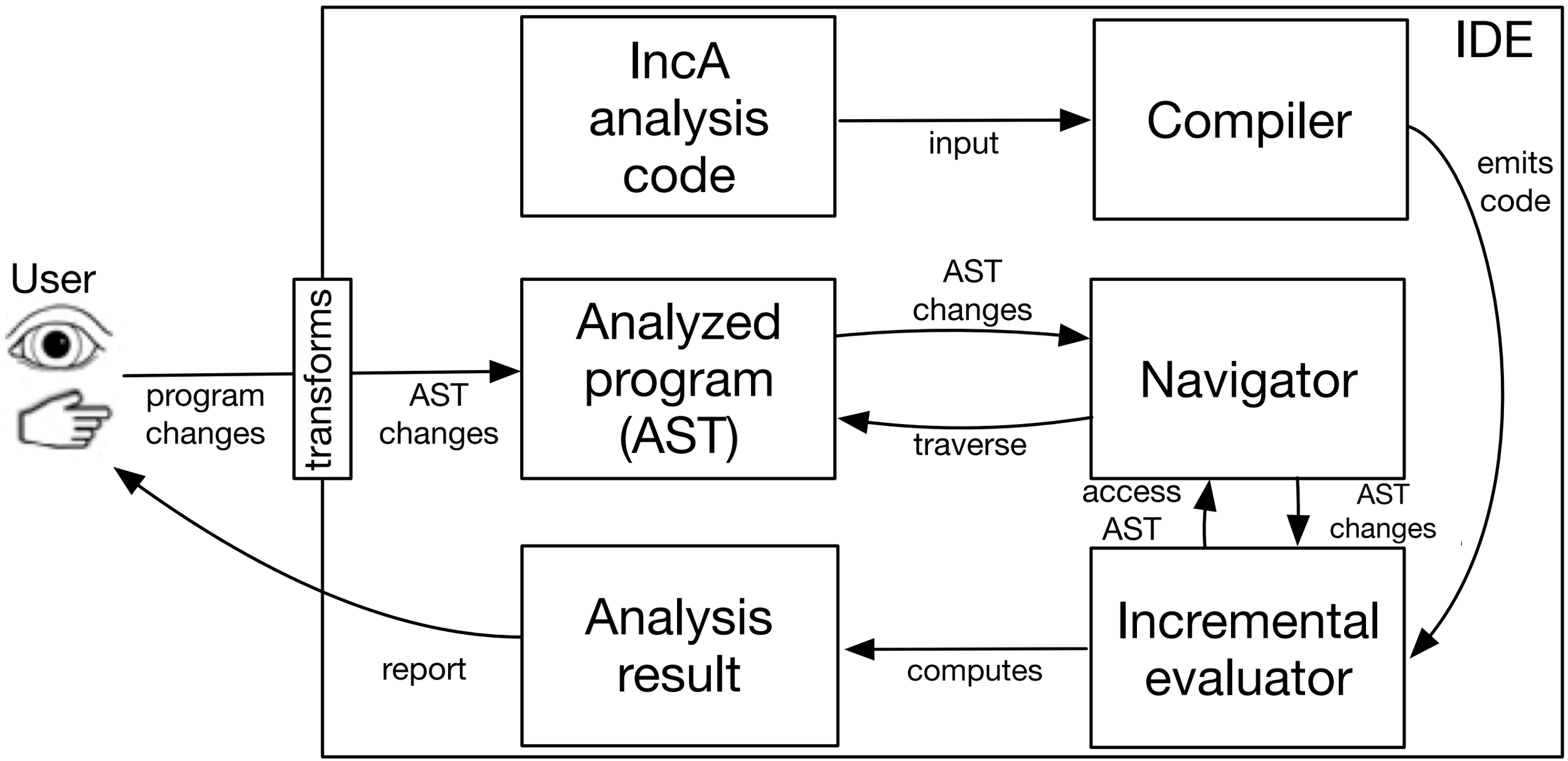
# IncA - Architecture



# IncA - Architecture



# IncA - Architecture



# IncA - Meta Analysis

Relational analysis over program elements of specific types.

CFlow ⊂ Stmt x Stmt

| | |
|---|---|
| 1 | 2 |
| 2 | 3 |
| 3 | 3a |
| 3 | 4 |
| 3a | 4 |

2. Many changes are irrelevant.

3. If a change is irrelevant, skip computation network.

4. Both run time and memory saving!

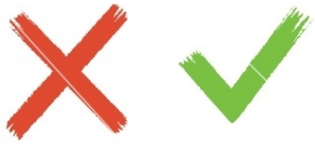
5. Meta Analysis: analyze the analysis code.

6. Compute the relevant types and use them as a hint.

# IncA - Meta Analysis

1. Relational analysis over program elements of specific types.

Many changes are irrelevant.



4. Both run time and memory saving!
5. Meta Analysis: analyze the analysis code.
6. Compute the relevant types and use them as a hint.

# IncA - Meta Analysis

Example: compute points-to targets (a = &b)

```
def pointsTo(s : Assignment) : (Var, Var) = {
  left := s.left
  right := s.right
  assert right instanceOf AddressOfExpr
  from := varInExpr(left)
  to := varInExpr(right.expr)
  return (from, to)
}
```

# IncA - Meta Analysis

Example: compute points-to targets (a = &b)

```
def pointsTo(s : Assignment) : (Var, Var) = {
  left := s.left
  right := s.right
  assert right instanceOf AddressOfExpr
  from := varInExpr(left)
  to := varInExpr(right.expr)
  return (from, to)
}
```

a = 10 ✗

a = &b ✓

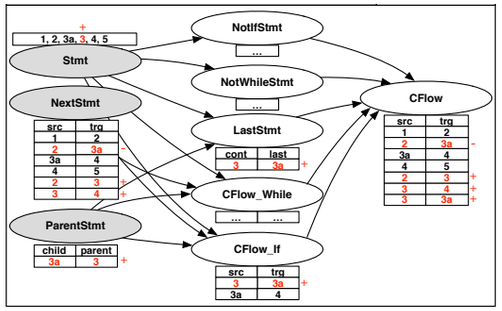
a = b + c ✗

# IncA - Meta Analysis

1. Relational analysis over program elements of specific types.
2. Many changes are irrelevant.

If a change is irrelevant,

skip computation network.




5. Meta Analysis: analyze the analysis code.
6. Compute the relevant types and use them as a hint.

# IncA - Meta Analysis

1. Relational analysis over program elements of specific types.
2. Many changes are irrelevant.
3. If a change is irrelevant, skip computation network.

Both run time and memory saving!

6. Compute the relevant types and use them as a hint.

# IncA - Meta Analysis

1. Relational analysis over program elements of specific types.
2. Many changes are irrelevant.
3. If a change is irrelevant, skip computation network.
4. Both run time and memory saving!

Analyze the analysis code.

# IncA - Meta Analysis

1. Relational analysis over program elements of specific types.
2. Many changes are irrelevant.
3. If a change is irrelevant, skip computation network.
4. Both run time and memory saving!
5. Meta Analysis: analyze the analysis code.

Compute the relevant types

and use them as a hint.

# IncA - Meta Analysis

1. Relational analysis over program elements of specific types.
2. Many changes are irrelevant.
3. If a change is irrelevant, skip computation network.
4. Both run time and memory saving!
5. Meta Analysis: analyze the analysis code.
6. Compute the relevant types and use them as a hint.

# IncA - Meta Analysis

1. Relational analysis over program elements of specific types.

+1 Not possible without a restricted DSL.

5. Meta Analysis: analyze the analysis code.
6. Compute the relevant types and use them as a hint.

# IncA Evaluation

Case study:

- Strong-update points-to analysis

# IncA Evaluation

Case study:

- Strong-update points-to analysis

| Project | Size (KLOC) | Description |
|---|---|---|
| Google Truth | 9 | an assertion framework |
| Google Gson | 14 | JSON serialization library |
| PGSQL JDBC | 45 | PostgreSQL-Java binding |
| Berkeley DB | 70 | an embedded database |

# IncA Evaluation

Case study:

- Strong-update points-to analysis

Two performance characteristics measured:

- Incremental runtime performance
- Memory consumption

# IncA Evaluation

Case study:

- Strong-update points-to analysis

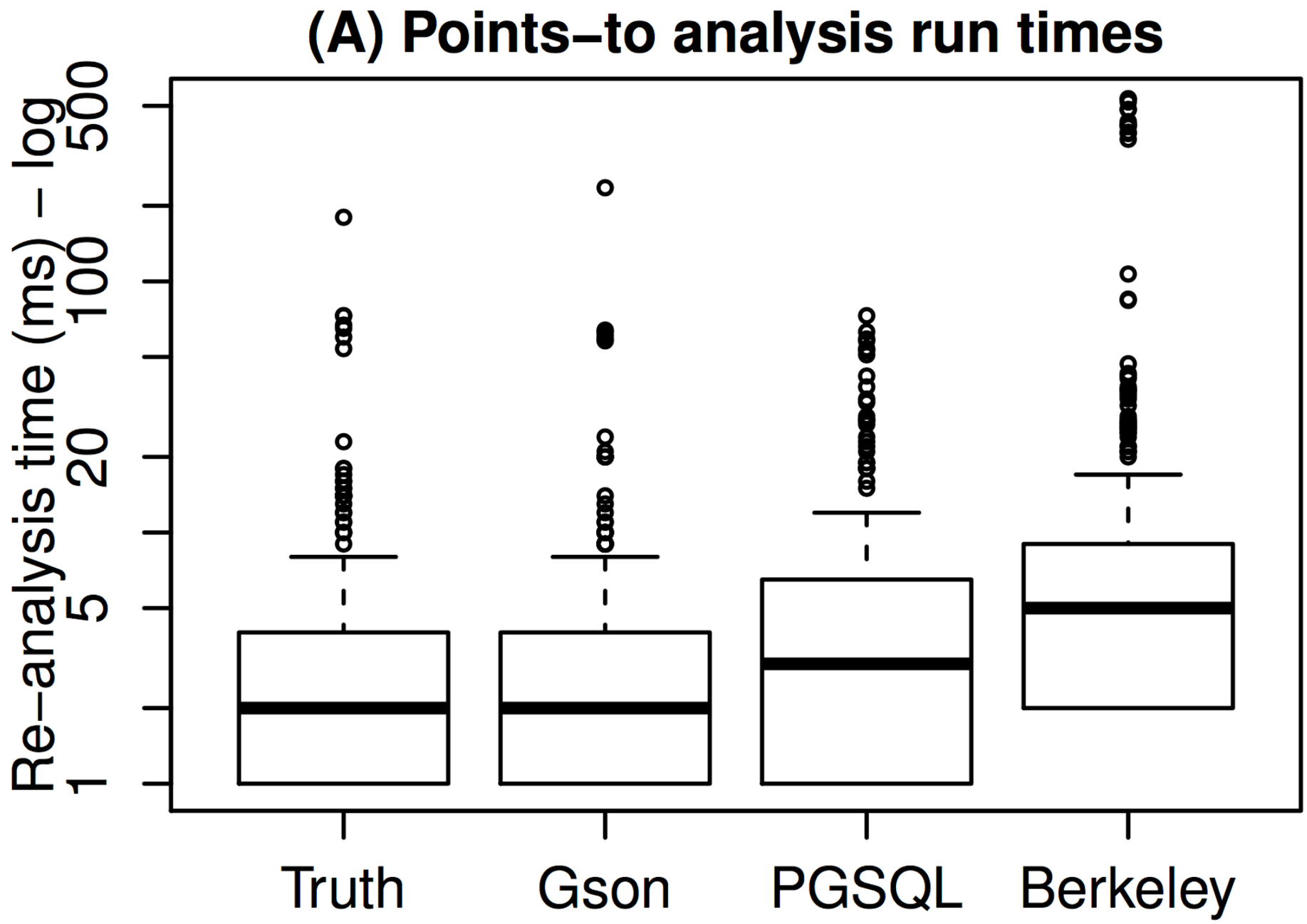
Two performance characteristics measured:

- Incremental runtime performance
- Memory consumption

Two kinds of incremental program changes

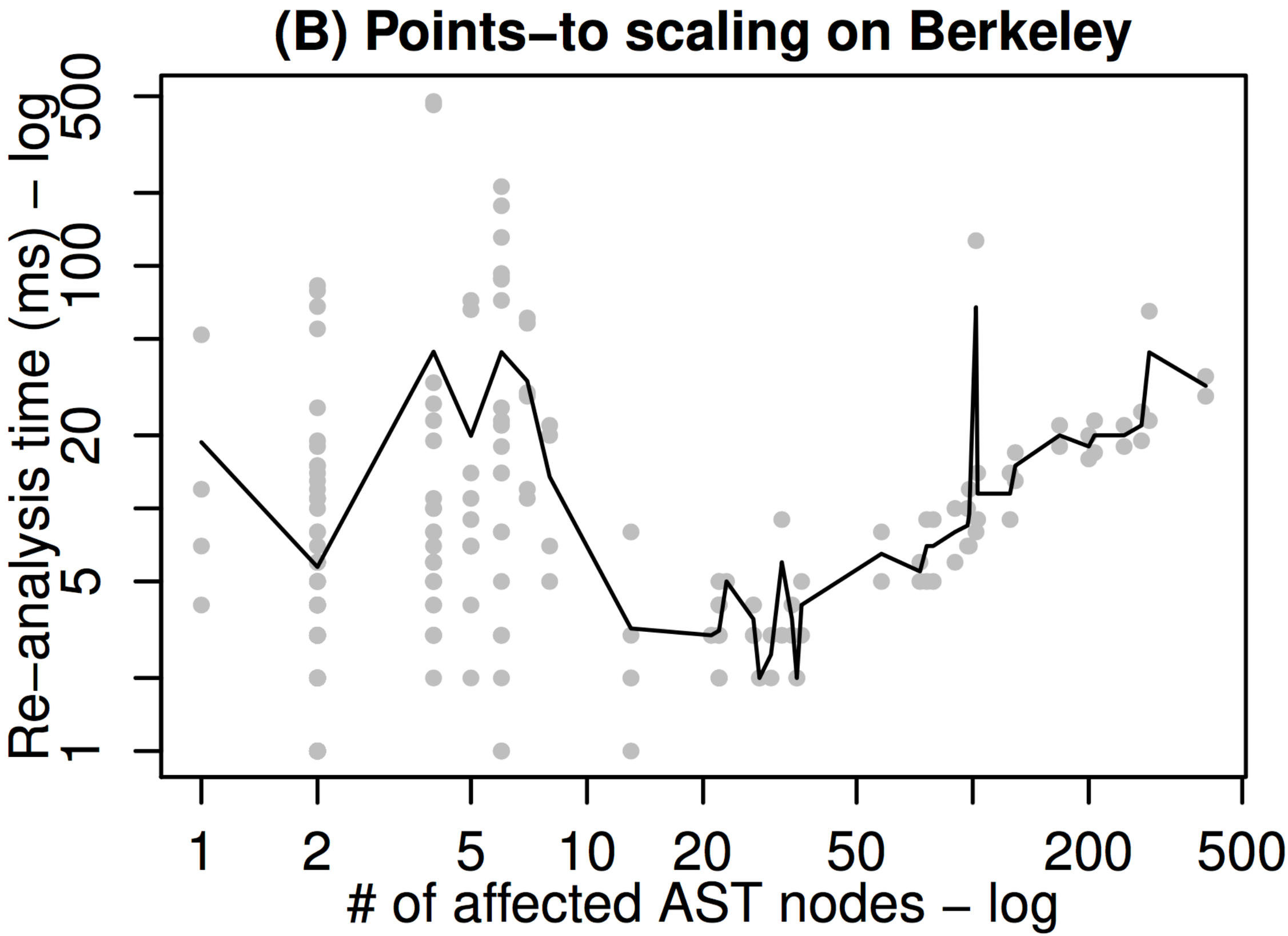
- Generic changes
- Changing assignments to directly exercise the analysis

# IncA Evaluation




| | Truth | Gson | PGSQL | Berkeley |
|---|---|---|---|---|
| Initialization | 6.8s | 9.6s | 60.5s | 63.4s |
| Flix | timed out on all four (5 min lim.) | | | |

# IncA Evaluation



# IncA Evaluation

Points-to (FS)

| Truth (9) | Gson (14) | PGSQL (45) | Berkeley (70) |
|---|---|---|---|
| 670 MB | 1 GB | 4.5 GB | 5 GB |

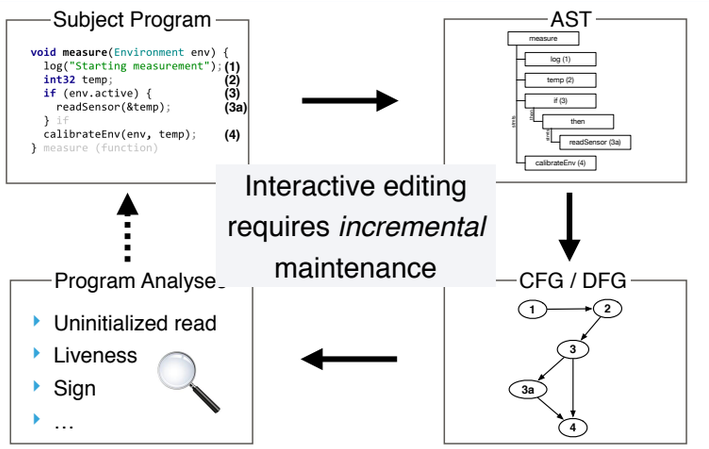
- MPS uses around ~2 GB
- Points-to analysis is flow-sensitive
- Flix points-to analysis uses 1 GB on 1 KLOC
- Future work: reduce mem. consumption with BDD or tuning of cache granularity

# Conclusions

| | |
|---|---|
| RUNTIME | The few ms incremental update times are exactly what interactive applications in IDEs need. |
| MEMORY | We pay the price with memory. It can grow large but not prohibitive. Concrete ideas for next steps. |
| EXPR. POWER | IncA supports static program analyses with user-defined lattices (~ “data flow analyses”). |

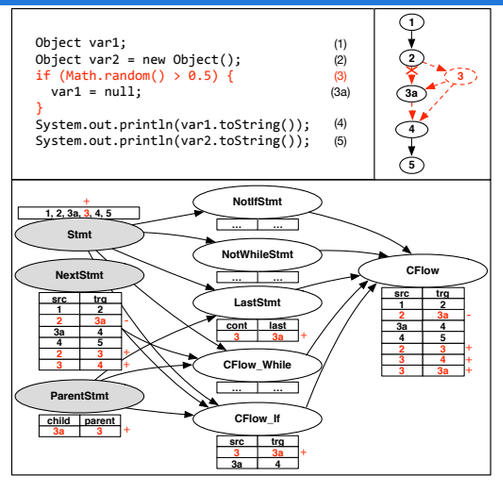
# Conclusions

## Analyses are essential in IDEs

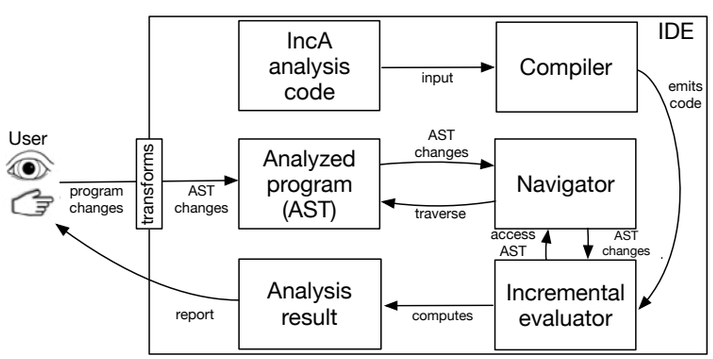



## Uninitialized Read Analysis

## IncA Evaluation




## IncA - Architecture




## IncA - Meta Analysis

Example: compute points-to targets (a = &b)

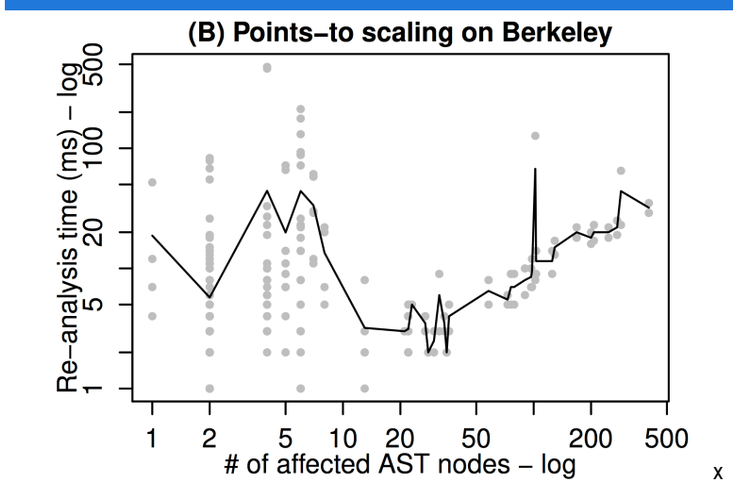
```
module pointsTo {
  def pointsTo(s : Assignment) : (Var, Var) = {
    left := s.left
    right := s.right
    assert right instanceOf AddressOfExpr
    from := varInExpr(left)
    to := varInExpr(right.expr)
    return (from, to)
  }
  private def varInExpr(e : Expression) : Var ={
    assert e instanceOf GlobalVarRef
    return e.var
  } alt {
    assert e instanceOf LocalVarRef
    return e.var
  }
}
```

right ∈ AddressOfExpr

left ∈ GlobalVarRef or LocalVarRef
from ∈ GlobalVar or LocalVar

right.expr ∈ GlobalVarRef or LocalVarRef
to ∈ GlobalVar or LocalVar

## IncA Evaluation




https://szabta89.github.io