Constraint Based Synthesis

Armando Solar-Lezama

MIT COMPUTER SCIENCE AND ARTIFICIAL INTELLIGENCE LABORATORY

Synthesis: 1980s view

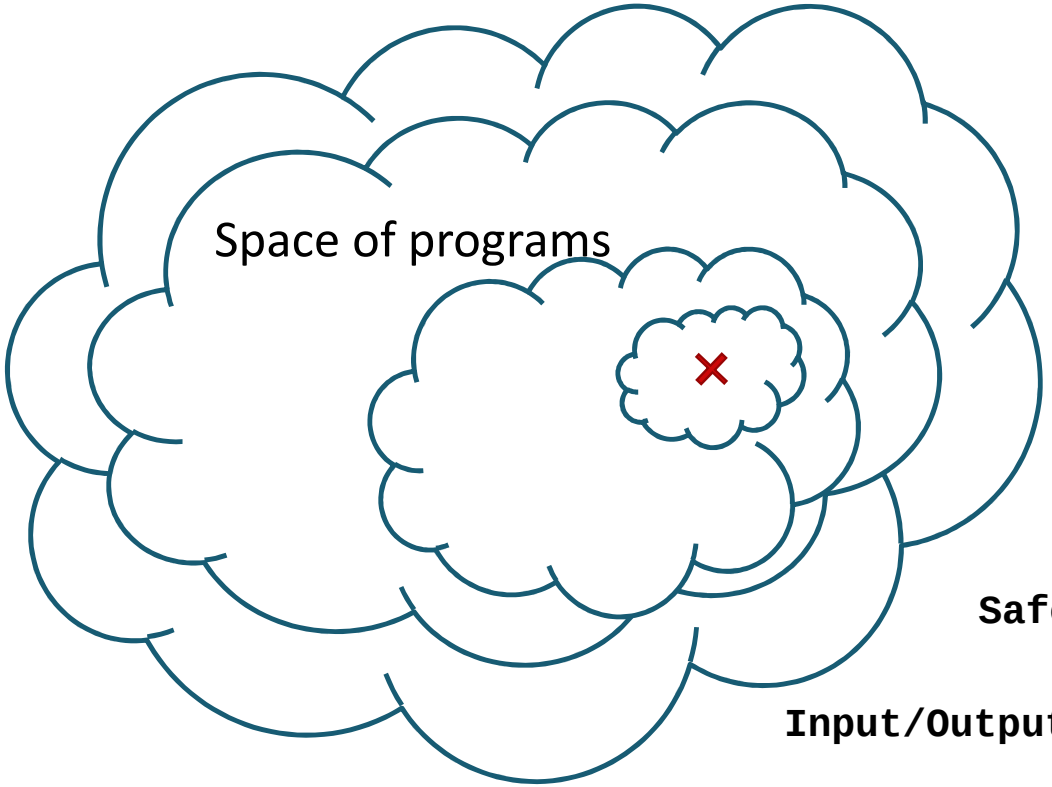
Complete Formal Specification

Theory $Sorting(\langle \alpha, \leq \rangle : linear - order)$
Imports $integer,\ bag(\alpha), seq(\alpha)$
Operations
$Ordered : seq(\alpha) \rightarrow Boolean$

Axioms
$\forall (S : seq(\alpha))\ (Ordered(S)$
$\Leftrightarrow \forall (i)(i \in \{1..length(S) - 1\} \implies S(i) \leq S(i+1)))$

```
int[N] sort(int N, int[N] input){
   int[N] output=input;
   int[N] done = 0;
   int k-0;
   for(int i=0; i<N; ++i){
      for(int j=i+1; j<N; ++j){
         if( output[j]< output[i]){
            int tmp = output[j];
            output[j] = output[i];
            output[i] - tmp;
         }
      }
   }
   return output;
}
```

Synthesis: modern view



P_c

$\varphi(c)$

Safety Properties

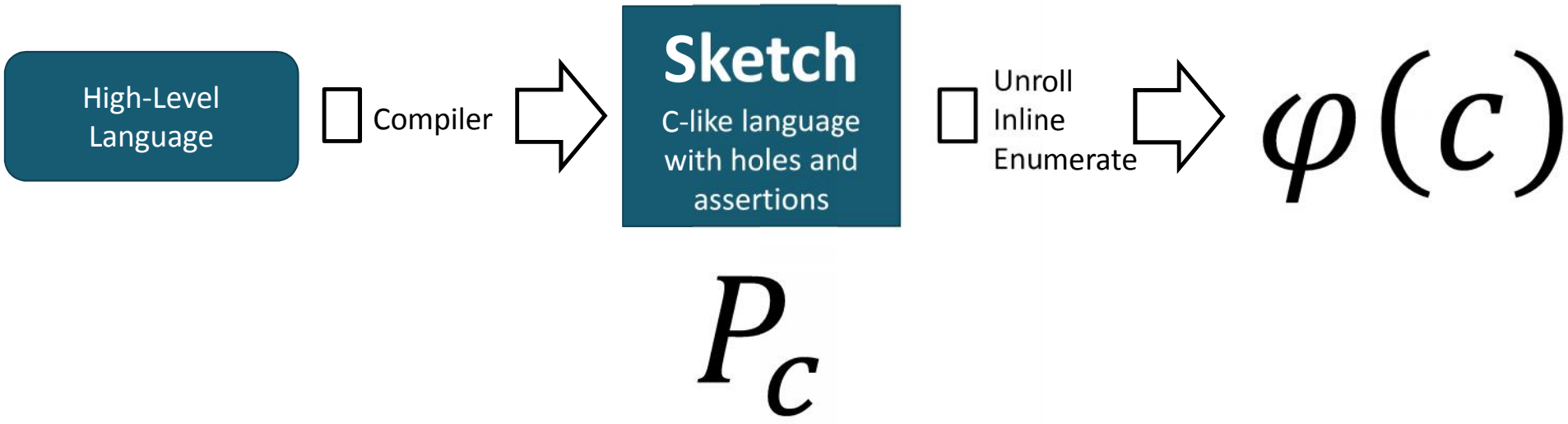
Input/Output Examples

Test Harnesses

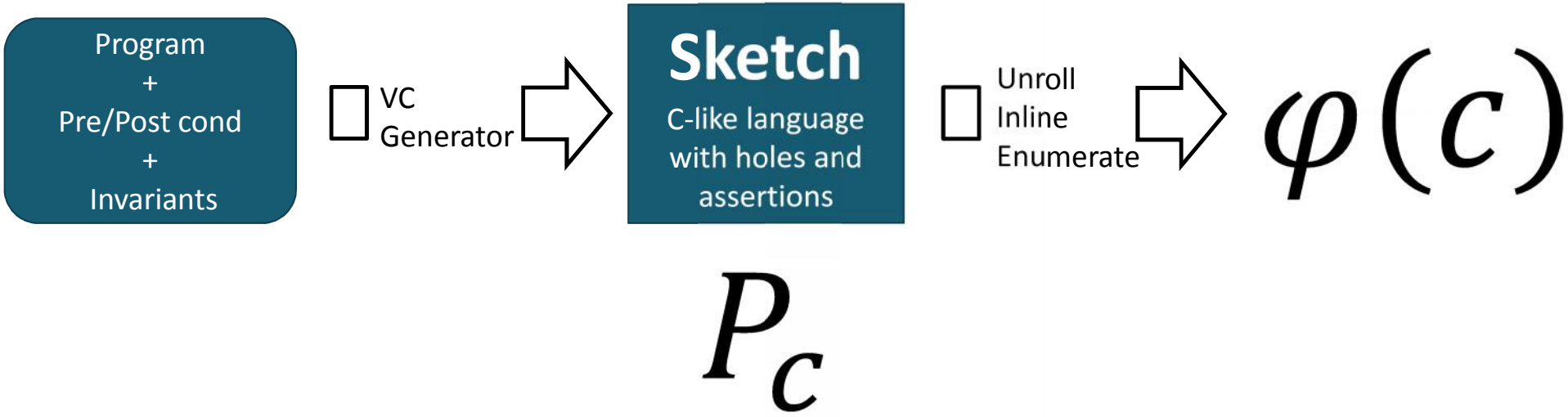
P_C

```c
/* Average of x and y without overflow */
int avg(int x, int y){
  int t =

  return t;
}
```

```
/* Average of x and y without overflow */
int avg(int x, int y){
  int t = y / 2 + x / 2 + ((x % 2 + y % 2) / 2);
  assert t == (x+y)/2;
  return t;
}
```



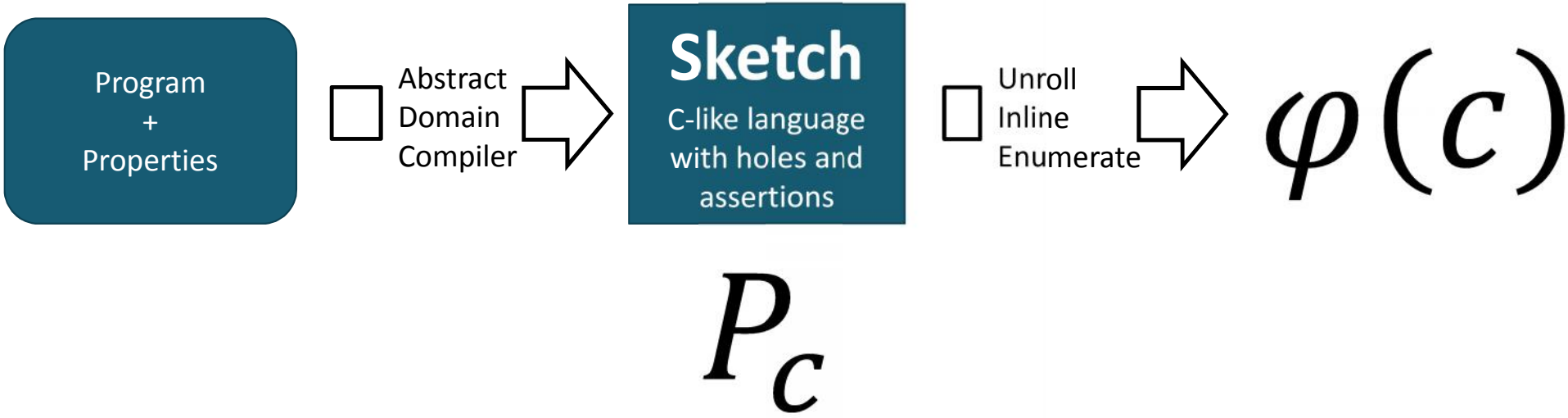
Automated Tutoring



Automated Tutoring

Provably Correct Synthesis

Program Optimization

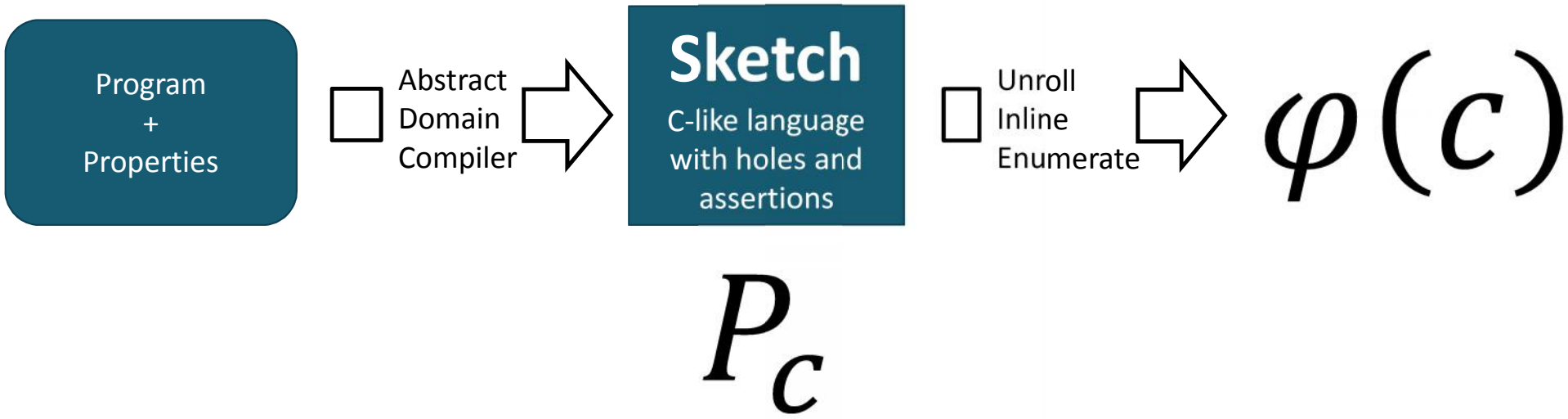


Automated Tutoring

Provably Correct Synthesis

Program Optimization

Visual Programming



Automated Tutoring

Provably Correct Synthesis

Program Optimization

Visual Programming

Automated Tutoring

With Rishabh Singh and Sumit Gulwani

Pinpoint student errors

```
def computeDeriv(poly):
    deriv = []
    zero = 0
    if (len(poly) == 1):
        return deriv
    for e in range(0, len(poly)):
        if (poly[e] == 0):
            zero += 1
        else:
            deriv.append(poly[e]*e)
    return deriv
```

replace derive by [0]

Should be a 1

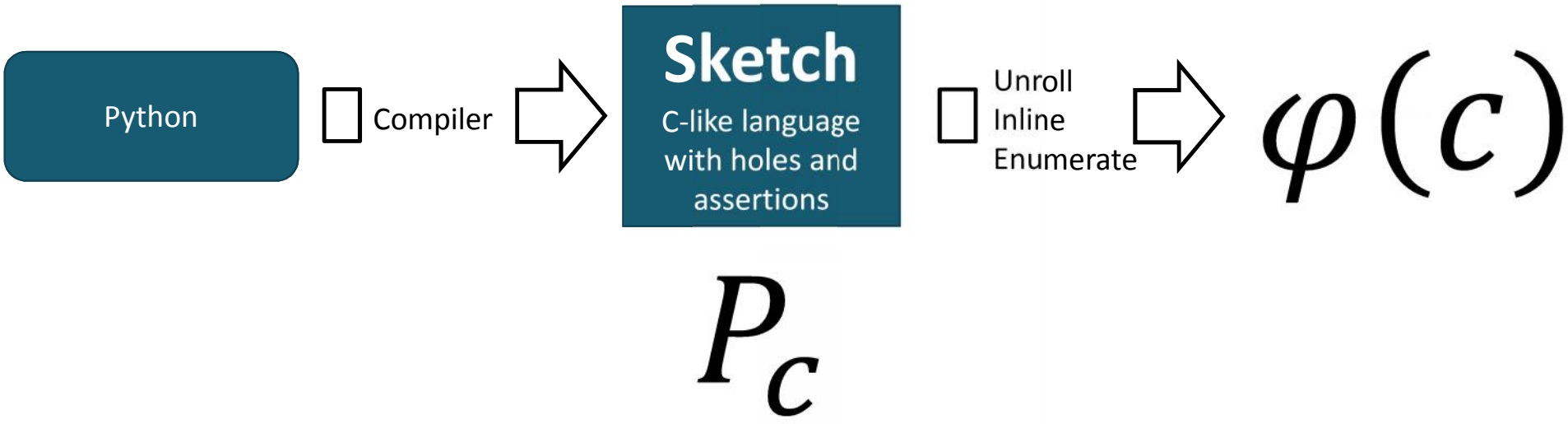
Get rid of this

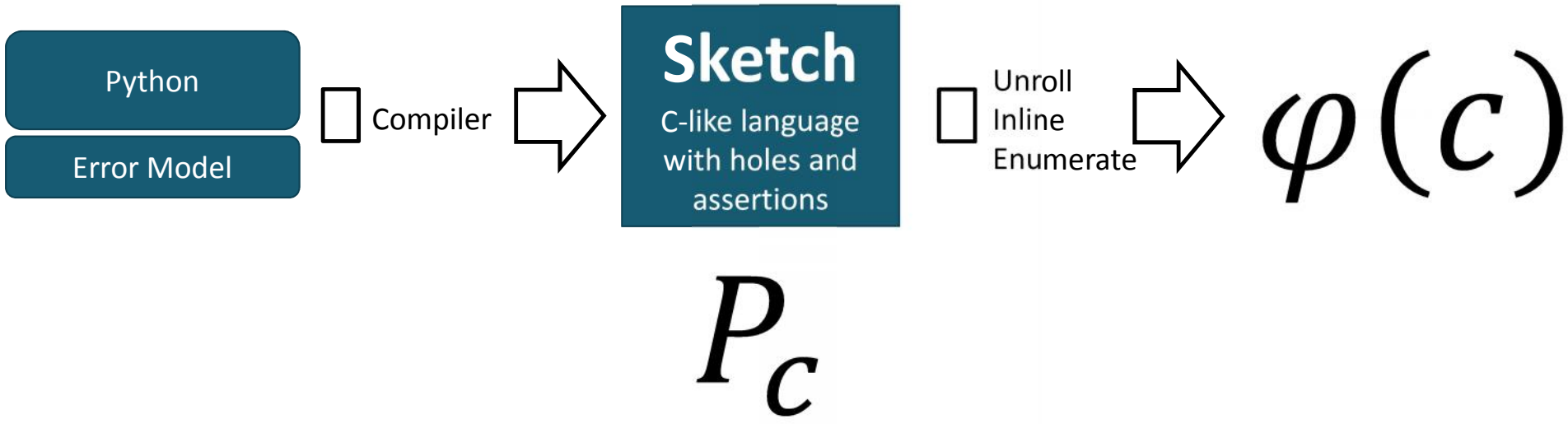
```
def computeDeriv(poly):
    length = int(len(poly)-1)
    i = length
    deriv = range(1,length)

    if len(poly) == 1:
        deriv = [0.0]
    else:
        while i >= 0:
            new = round((poly[i] * i),2)
            i -= 1
            deriv[i] = new

    return deriv
```

Should be a 0

Should be >

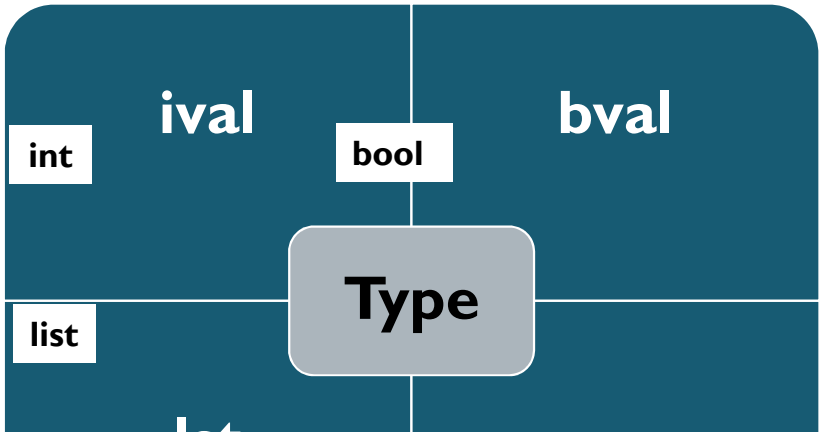
This is a synthesis problem

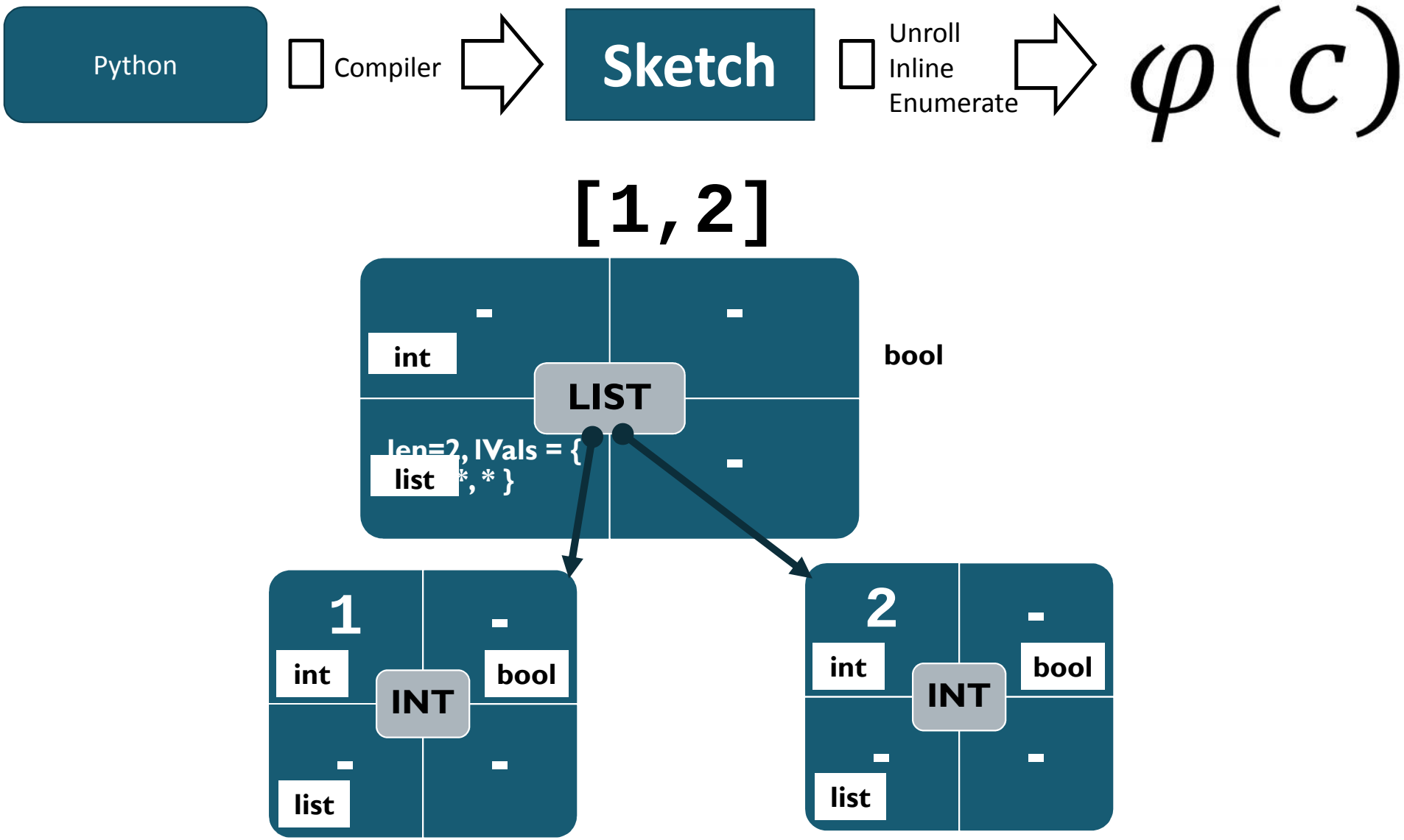

Python
Compiler
Sketch
C-like language with holes and assertions
Unroll
Inline
Enumerate
$\varphi(c)$
P_c


Python
Error Model
Compiler
Sketch
C-like language with holes and assertions
Unroll
Inline
Enumerate
$\varphi(c)$
P_c

Dynamic Types

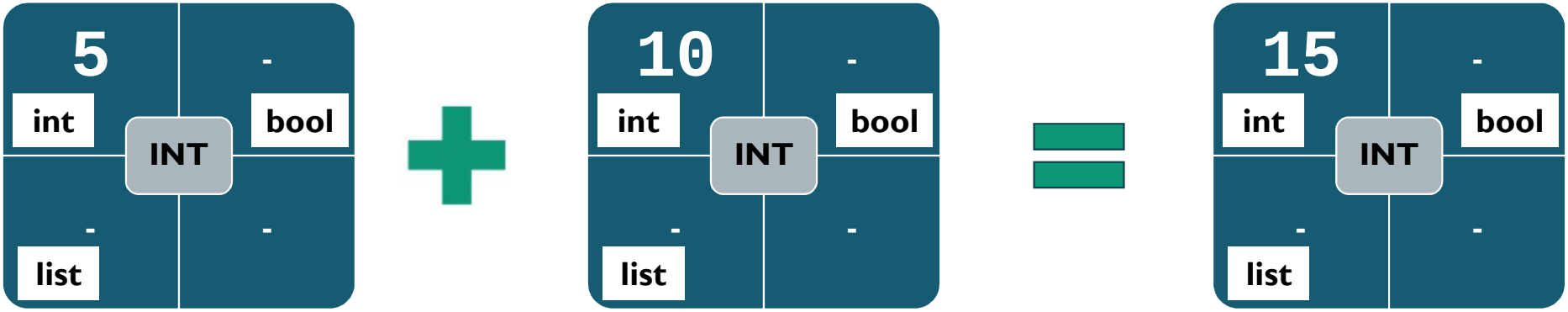
```
struct MultiType{
  int type;
  int ival;
  bit bval;
  MTString str; MTList lst;
  MTDict dict; MTTuple tup;
}
```




Python
Compiler
Sketch
Unroll
Inline
Enumerate
$\varphi(c)$
[1,2]
int
LIST
bool
list
1
int
INT
bool
list
2
int
INT
bool
list

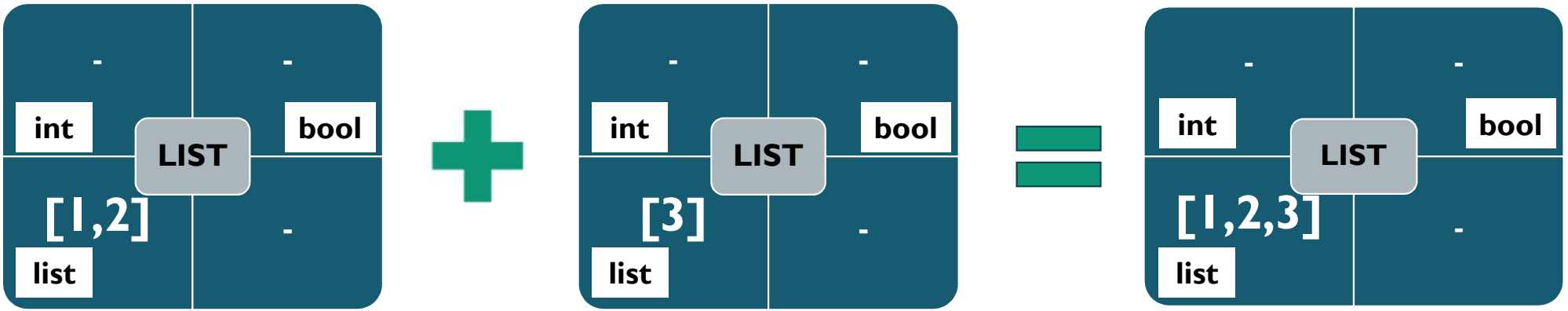

Python
Compiler
Sketch
Unroll
Inline
Enumerate
$\varphi(c)$

```
x + y → addMT(x,y)
```


5
int
INT
bool
list
10
int
INT
bool
list
15
int
INT
bool
list

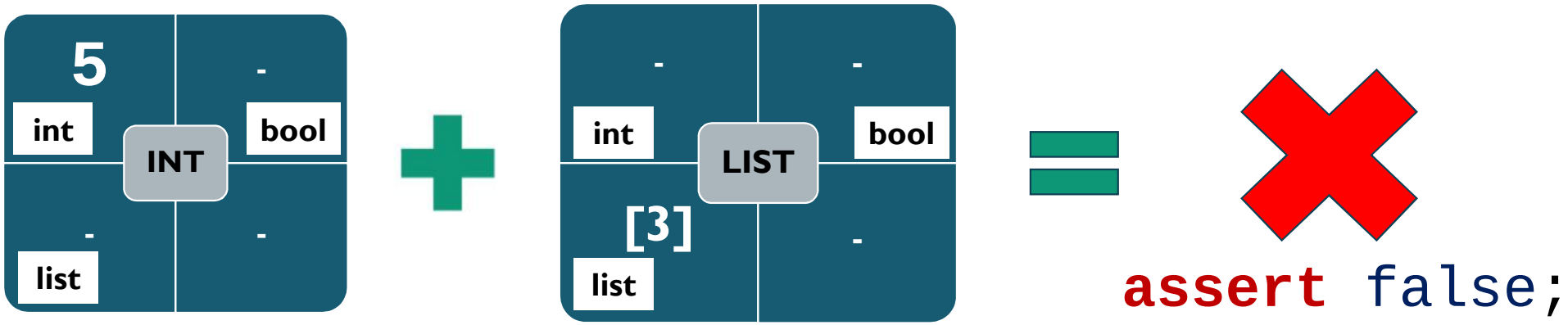

Python
Compiler
Sketch
Unroll
Inline
Enumerate
$\varphi(c)$

```
x + y → addMT(x,y)
```


-
-
int
LIST
bool
[1,2]
-
list
-
-
int
LIST
bool
[3]
-
list
-
-
int
LIST
bool
[1,2,3]
-
list
-

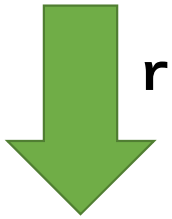

Python
Compiler
Sketch
Unroll
Inline
Enumerate
$\varphi(c)$

```
x + y → addMT(x,y)
```


5
-
int
bool
INT
-
-
list
-
-
int
bool
LIST
[3]
-
list
assert false;

- `return a` → `return [0]`
- `range(`a_1`, `a_2`)` → `range(`a_1`+1,`a_2`)`
- a_0 `==` a_1 → `False`


```
range(0, len(poly))
```



$\texttt{range}(a_1, a_2) \rightarrow \texttt{range}(a_1+1, a_2)$

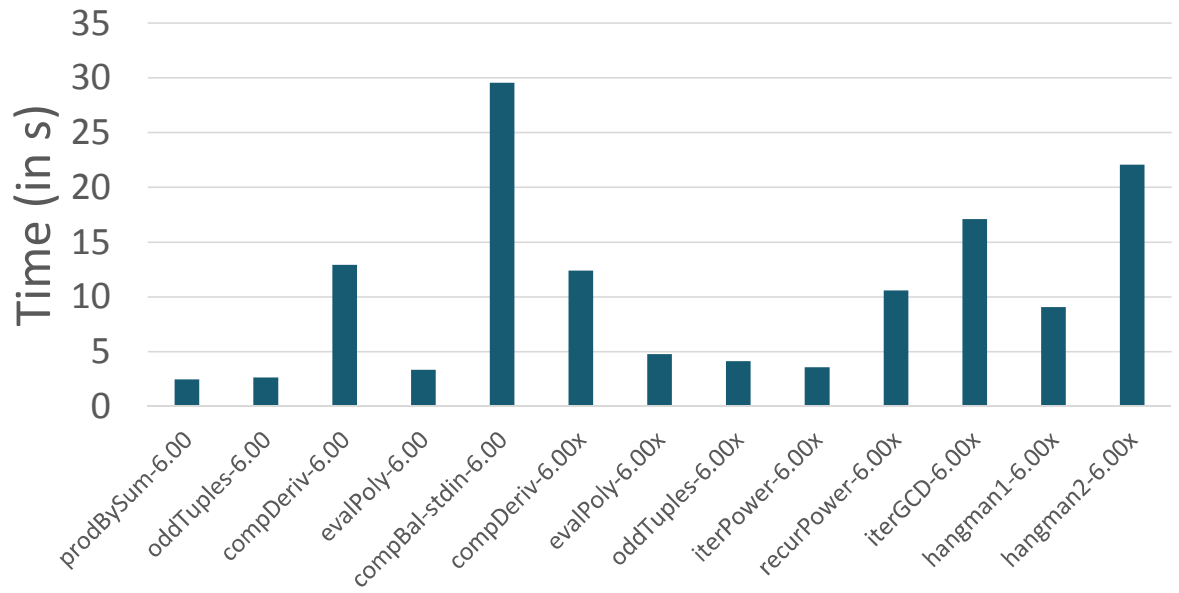
```
range({0 ,1}, len(poly))
```

default choice

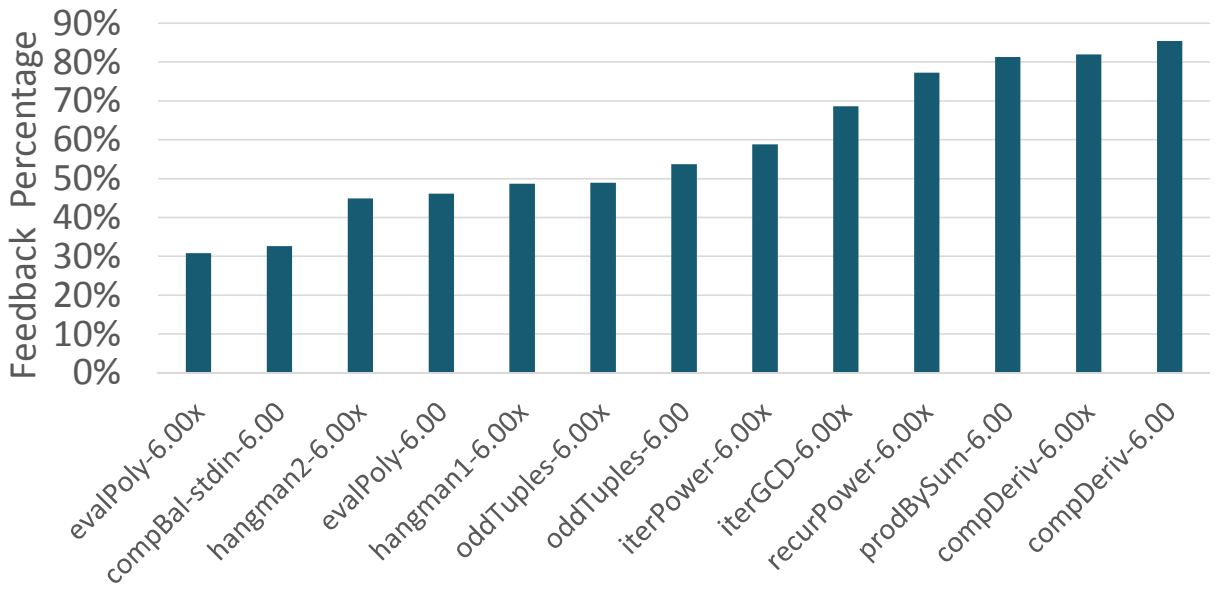
{ , } → modifyMT0( )

```
MultiType modifyMT0( ){
    if(??) return    // default choice
    else
        choice0 = true
        return                  // non-default choice
}
```

Benchmark	Test Set
evalPoly-6.00	13
compBal-stdin-6.00	52
compDeriv-6.00	103
hangman2-6.00x	218
prodBySum-6.00	268
oddTuples-6.00	344
hangman1-6.00x	351
evalPoly-6.00x	541
compDeriv-6.00x	918
oddTuples-6.00x	1756
iterPower-6.00x	2875
recurPower-6.00x	2938
iterGCD-6.00x	2988



Average Running Time (in s)



Feedback Generated (Percentage)

Invariant Synthesis for Optimization

Work with Alvin Cheung and Shoaib Kamil

Optimization then and now

Naïve source code

~~Optimal executable~~

Kind-of-OK executable

Domain specific problem description

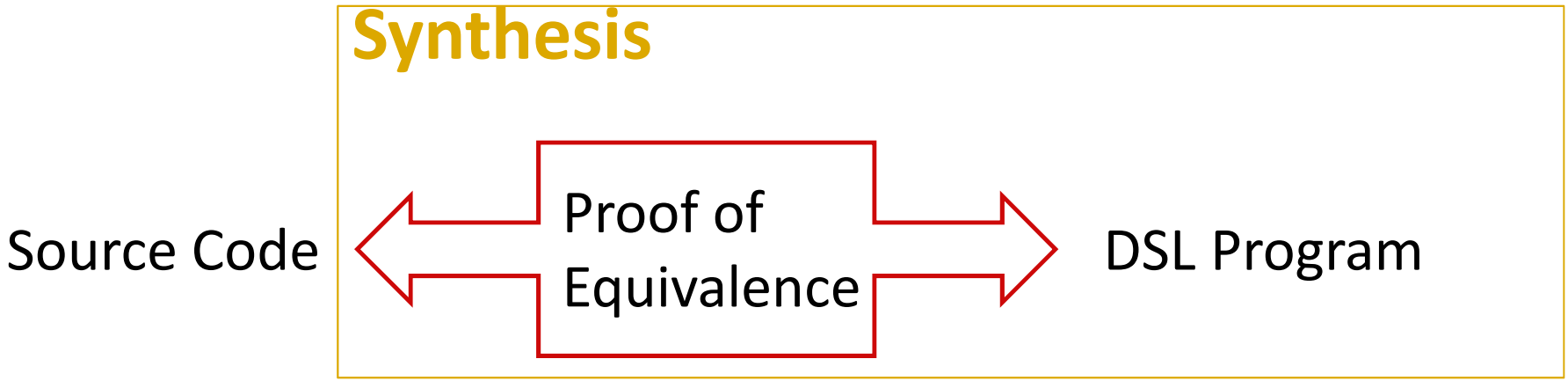
ATLAS

Pochoir

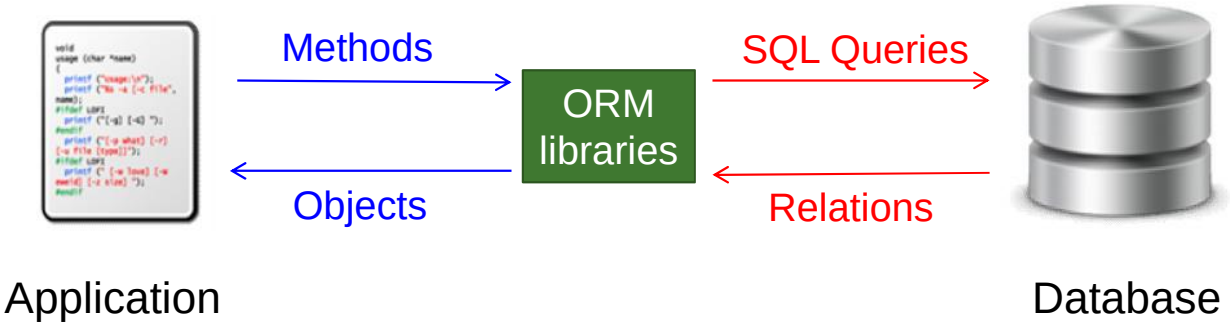
Halide

Close to optimal implementation

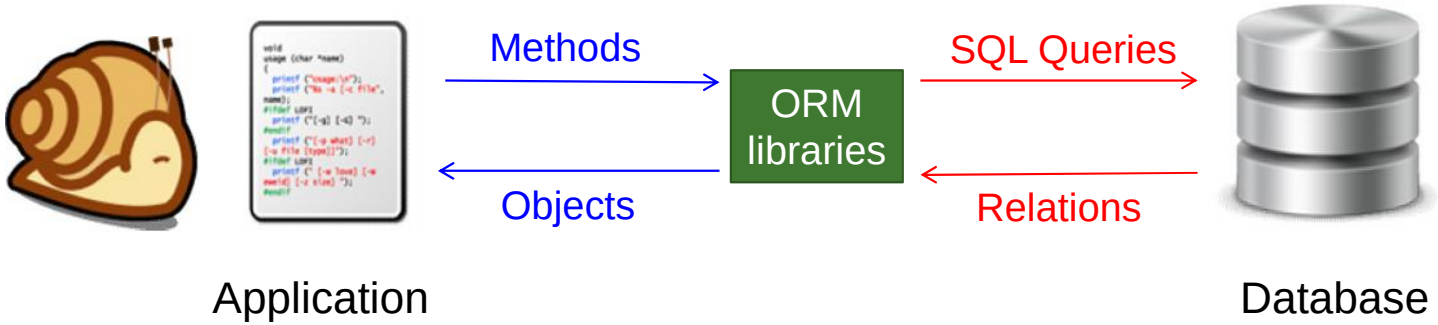
What about existing source code?



Java to SQL


Methods
ORM libraries
SQL Queries
Objects
Relations
Application
Database

Java to SQL


Methods
Objects
ORM libraries
SQL Queries
Relations
Application
Database

Java to SQL

```
List getUsersWithRoles () {
  List users = User.getAllUsers();
  List roles = Role.getAllRoles();
  List results = new ArrayList();
  for (User u : users) {
       for (Role r : roles) {
         if (u.roleId == r.id)
            results.add(u); }}
  return results; }
```

SELECT * FROM user

SELECT * FROM role

convert to

```
List getUsersWithRoles () {
 return executeQuery(
  "SELECT   u FROM user u, role r   WHERE    u.roleId == r.id
     ORDER BY u.roleId, r.id"; }
```

Java to SQL

```
List getUsersWithRoles () {
  List users = User.getAllUsers();
  List roles = Role.getAllRoles();
  List results = new ArrayList();
  for (User u : users) {
       for (Role r : roles) {
         if (u.roleId == r.id)
            results.add(u); }}
  return results; }
```

outerInvariant(users, roles, u, results, …)

innerInvariant(users, roles, u, r, results, …)

results = outputExpr(users, roles)

Verification conditions

preCondition →

outerInvariant(users/query(…), results/[], …)

outerInvariant(…) ∧ outer loop terminates →

results = outputExpr(users, roles) …


Program
+
Unkown Post cond
+
Unknown Invariants
VC
Generator
Sketch
C-like language
with holes and
assertions
Unroll
Inline
Enumerate
$\varphi(c)$

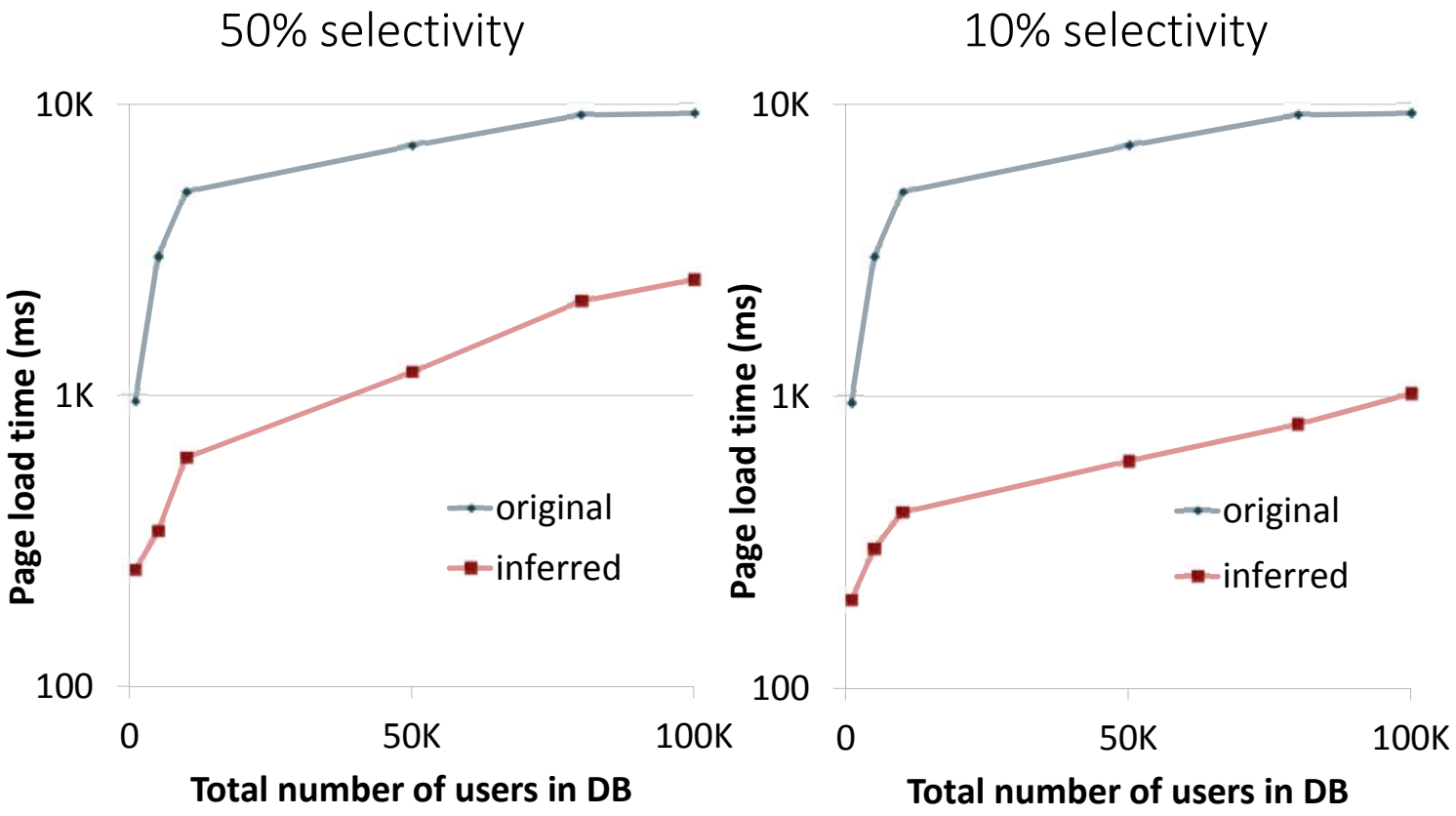
Real-world Evaluation

Wilos (project management application) – 62k LOC		
Operation type	# Fragments found	# Fragments converted
Projection	1	1
Selection	13	10
Join	7	7
Aggregation	11	10
Total	**33**	**28**

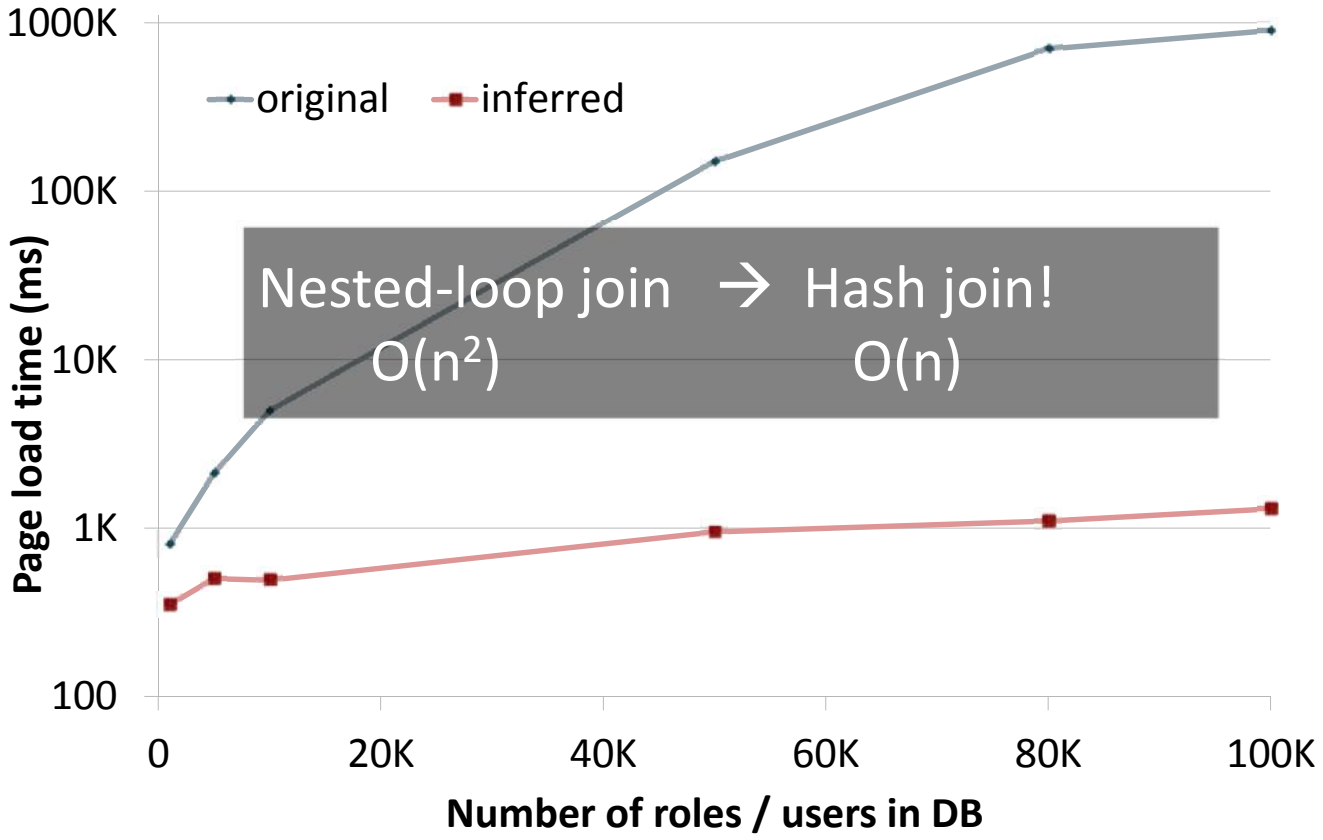
Real-world Evaluation

iTracker (bug tracking system) – 61k LOC		
Operation type	# Fragments found	# Fragments converted
Projection	3	2
Selection	3	2
Join	1	1
Aggregation	9	7
Total	**16**	**12**

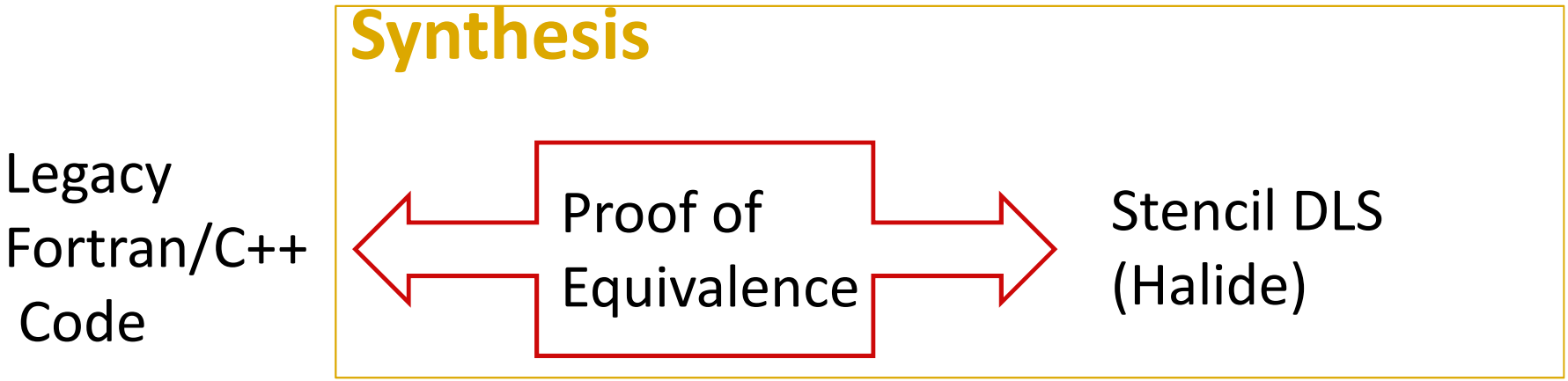
Selection Query



Join Query


1000K
100K
10K
1K
100
Page load time (ms)
original
inferred
Nested-loop join → Hash join!
O(n²) O(n)
0
20K
40K
60K
80K
100K
Number of roles / users in DB

What about HPC?



Legacy to Halide

```
for (k=y_min-2;k<=y_max+2;k++) {
   for (j=x_min-2;j<=x_max+2;j++) {
      post_vol[((x_max+5)*(k-(y_min-2))+(j)-(x_min-2))]
          =volume[((x_max+4)*(k-(y_min-2))+(j)-(x_min-2))]
             + vol_flux_y[((x_max+4)*(k+1 -(y_min-2))+(j)-(x_min-2))]
             - vol_flux_y[((x_max+4)*(k-(y_min-2))+(j)-(x_min-2))];
   }
}
```

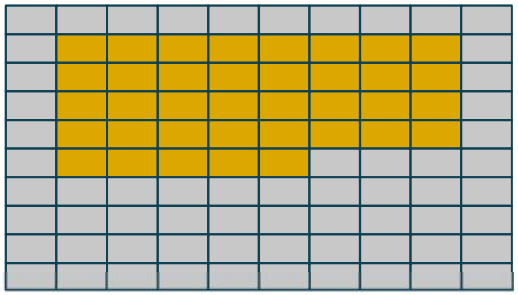
$\forall (j,k) \in Dom$ `post_vol[j,k] = volume[j,k]`
`+ vol_flux[j, k+1] + vol_flux[j, k]`

Invariants

$$\forall\ (i,j) \in Dom:$$
$$A[i,j] = Expr(\ \{B_n[expr(i,j), expr(i,j)]\ \})$$

Invariants

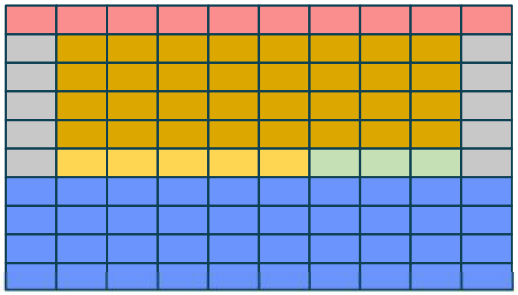
```
for(int i=0; i<n-1; ++i)
  for(int j=0; j<m-1; ++j)
     out[i,j] = sin(in[i,j])
```



$$\forall\,(i,j) \in Dom:$$
$$A[i,j] = Expr(\ \{B_n[expr(i,j), expr(i,j)]\}\ )$$

Invariants

```
for(int i=0; i<n-1; ++i)
  for(int j=0; j<m-1; ++j)
     out[i,j] = sin(in[i,j])
```



$$\forall\,(i,j) \in Dom:$$
$$A[i,j] = Expr(\ \{B_n[expr(i,j), expr(i,j)]\}\ )$$

Challenges

- Big invariants
- Complex floating point arithmetic
- Universal Quantifiers

Quantifiers

$$\exists c \; \forall in \; P_c(in)$$

```
if(outerInvariant(…) && cond()){
        assert  out == outputExpr(in) ;
}
```

outerInvariant(…) ∧ outer loop terminates → output= outputExpr(in)

The ∀ leads to a ∃∀∃ constraint!

Quantifiers

$$\exists c \; \forall in \; P_c(in)$$

Always safe to weaken this condition (more true)

```
if(outerInvariant(…) && cond()){
        assert  out == outputExpr(in)  ;
}
```

$$\bigwedge_{(i,j)\in E} out[i,j] = Expr(\ \{in[expr(i,j), expr(i,j)]\}\ )$$

Let the synthesizer discover E!

Does it work?

- Can compute invariants for all stencils in Cloverleaf
- Most complex one takes 20 hrs on 15 Cores
- Currently pushing new benchmarks

- There is more to synthesis than “synthesis”
- You can do this too!
  - All the sketch infrastructure is available in open source