Review

Annotated Program

```
method Eg2 (x, y : int)
  returns (z : int)
{
   assume x < y;
   z := (x+y)/2;
   assert x < z;
}
```

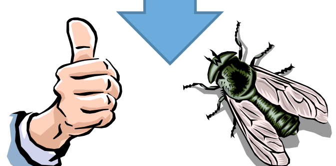
Verification Condition Generation

Verification Condition

Valid iff for all x, y, z:
$((x<y) \wedge (z=x+y/2) \Rightarrow (x<z))$

Foundation: Mathematical Logic

Theorem Prover/ SAT/SMT Solver



SAT/SMT Solvers

- SMT : Satisfiability Modulo Theories

$$\neg((x<y)\wedge(z=x+y/2)\Rightarrow(x<z))$$

SMT Instance

$$\neg(p(x,y)\wedge(z=g(f(x,y),2\downarrow c)\Rightarrow p(x,z))$$

SAT Instance

SMT Solvers

$$\left[\ \phi(1) \land \forall n.\ \phi(n) \Rightarrow \phi(n+1)\right]$$

- Combine
  - SAT Solvers +
  - Specialized Solvers for specific theories

$$\Rightarrow (\forall n.\ \phi(n))$$

- Theories are characterized by
  - A set of axioms or axiom schemas
  - An axiom schema corresponds to a potentially infinite set of axioms
    - E.g., principle of mathematical induction
    - Must be recursively enumerable

The Z3 Theorem Prover

- http://rise4fun.com/Z3/tutorial/guide

```
method Eg2 (x, y : int)
  returns (z : int)
{
   assume x < y;
   z := (x + y) / 2;
   assert x < z;
}
```

$$\neg((x<y)\wedge(z=x+y/2)\Rightarrow(x<z))$$

A puzzle (from BrainBashers.com)

During a recent police investigation, Chief Inspector Stone was interviewing five local villains to try and identify who stole Mrs Archer's cake from the mid-summers fayre. Below is a summary of their statements:

```
Arnold:   it wasn't Edward
          it was Brian

Brian:    it wasn't Charles
          it wasn't Edward

Charles:  it was Edward
          it wasn't Arnold

Derek:    it was Charles
          it was Brian

Edward:   it was Derek
          it wasn't Arnold
```

$\neg e$
b

$[(\neg e \wedge \neg b) \vee (e \wedge b)] \wedge$

$[\quad] \wedge$

$[\quad] \wedge (a \vee b \vee c \vee d \vee e)$
$\wedge (a \Rightarrow \neg b \wedge \neg c \wedge \neg d \ldots)$

It was well known that each suspect told exactly one lie. Can you determine who stole the cake?

- Encode the puzzle as a SAT problem
- How do *<u>you</u>* solve the puzzle?

Towards Automated Verification

Annotated Program

```
method Eg2 (x, y : int)
  returns (z : int)
{
   assume x < y;
   z := (x+y)/2;
   assert x < z;
}
```

Verification Condition Generation

Verification Condition

Valid iff for all x, y, z:
$((x<y) \wedge (z=x+y/2) \Rightarrow (x<z))$

Theorem Prover/ SAT/SMT Solver

Foundation: Hoare Logic

Foundation: Mathematical Logic

Hoare Triples

- Syntax: $\{P\}\ S\ \{Q\}$
  - Where P and Q are predicates (assertions) and S is a statement (code fragment)

- Intended meaning: If statement S is executed starting in an initial state satisfying P and it terminates, the resulting final state will satisfy Q

Example

```
method Eg2 (x, y : int)
  returns (z : int)
{
    assume x < y;
    z := (x+y)/2;
    assert x < z;
}
```

$\{ x < y \}$
z := (x+y)/2;
$\{ x < z \}$

$\{ x < y \}$
z := (x+y)/2;
$\{ x \leq z \}$

Exercise

- Write a specification

- … for a program S that swaps the values of two variables x and y

$\{x = x_{orig} \wedge y = y_{orig}\}$

S
temp := x;
x := y;
y := temp;

$\{x = y_{orig} \wedge y = x_{orig}\}$

Exercise

- Is the following valid?

```
{ n > 0 }
sum := 0;
i := 0;
while (i < n) {
    sum := sum – 1;
    i := i – 1;
}
{ sum == n*(n+1)/2}
```

Program Correctness

- Partial correctness vs. total correctness
- Proving program termination usually harder and requires different techniques

Exercise

- What will really happen if we run this program (in practice today)?

```
{ n > 0 }
sum := 0;
i := 0;
while (i < n) {
      sum := sum – 1;
      i := i – 1;
}
{ sum == n*(n+1)/2}
```

Integers in practice

- Integer variables & arithmetic
  - Classical (standard) theory
  - Reality: 32-bit and 64-bit integers
  - Bit-vector arithmetic theory
  - Modular arithmetic

Vocabulary (Syntax)

$0, 1, +, -, \times, /, \leq$

Different interpretations (structures)

Classical Integers | 32-bit signed integers | 64-bit unsigned integers

$\{ x < y \}$

y := y+1;

$\{ x < y \}$

Arrays

- Specify that an array A is sorted

- How can we model arrays?

- The logic we have considered so far is untyped. We can extend this to a typed version (called many-sorted logic)
  - “sort” $\approx$ “type”

- We will stick to plain untyped logic here

The Array Datatype

- Function symbols used to model arrays
  - length (unary)
    - length(A)
    - We can use A.length as syntactic sugar
  - lookup (binary)
    - lookup(A,i)
    - We can use A[i] as syntactic sugar
  - update (ternary)
    - update(A,i,v)
    - We can use $A[i \leftarrow v]$ as syntactic sugar
    - same as array A except at index i where it contains the value v
  - isArray (unary)
    - type information

Exercise

- Specify that an array A is sorted

$\forall i.\ 0 \le i \le \mathrm{length}(A) - 2.$

$\Rightarrow A[i] \le A[i+1]$

Exercise

- Specify that a program S sorts an input array A

~~$\{A = A'\}$~~ $\{A = A'\}$

S

$\{\mathit{sorted}(A) \wedge \underbrace{\mathit{perm}(A, A')}\}$

Exercise

- What are appropriate axioms for the array datatype?

A_1: $\forall A\ \forall i\ \forall v.\ \text{isArray}(A) \Rightarrow$

$\text{lookup}(\text{update}(A, i, v), i) = v$

$i \neq j \Rightarrow$

A_2: . . . .

$\text{lookup}(\text{update}(A, i, v), j) =$

$\text{lookup}(A, j)$

Pointers & Heap

- Memory can be modeled as one (or more) giant array

- Naïve modeling may not be sufficient
  - Due to incompleteness

- Separation Logic
- Pointer analysis
- TVLA
  - An abstract-interpretation based approach to verification in the presence of dynamic structures

Hoare Logic

- Inference rules for proving $\vdash \{P\}\, S\, \{Q\}$
- Combines
  - Rules for standard mathematical logic with
  - Rules for reasoning about programming language constructs

Hoare Logic

Basic Constructs

- Assignment statement: $x := e$
- Statement sequencing: $S_1 ; S_2$
- Conditional: $if\ (E)\ then\ S_1\ \ else\ S_2$
- Iteration: $while\ (e)\ do\ S$

Inductive Definitions Of Sets and Relations

- Syntax

$$\phi ::= P \quad | \quad \phi{\downarrow}1 \vee \phi{\downarrow}2 \quad | \quad \phi{\downarrow}1 \wedge \phi{\downarrow}2$$

Antecedent

$$M \vDash \phi{\downarrow}1 \,, \qquad M \vDash \phi{\downarrow}2 \;/ M \vDash \phi{\downarrow}1 \wedge \phi{\downarrow}2$$

- Semantics

Consequent

$$\vdash \varphi{\downarrow}1 \,, \qquad \vdash \varphi{\downarrow}2 \;/ \vdash \varphi{\downarrow}3$$

- Proof rules

Statement Sequencing

$$?/\vdash \{P\}\, S_1 ; S_2 \; \{Q\}$$

$$\vdash \{P\}\, S_1 \; \{R\}, \quad \vdash \{R\}\, S_2 \; \{Q\} / \vdash \{P\} S_1 ; S_2 \, \{Q\}$$

Conditional Statement

$$? / \vdash \{P\}\ \text{if } (E) \text{ then } S{\downarrow}1 \ \text{ else } S{\downarrow}2 \ \{Q\}$$

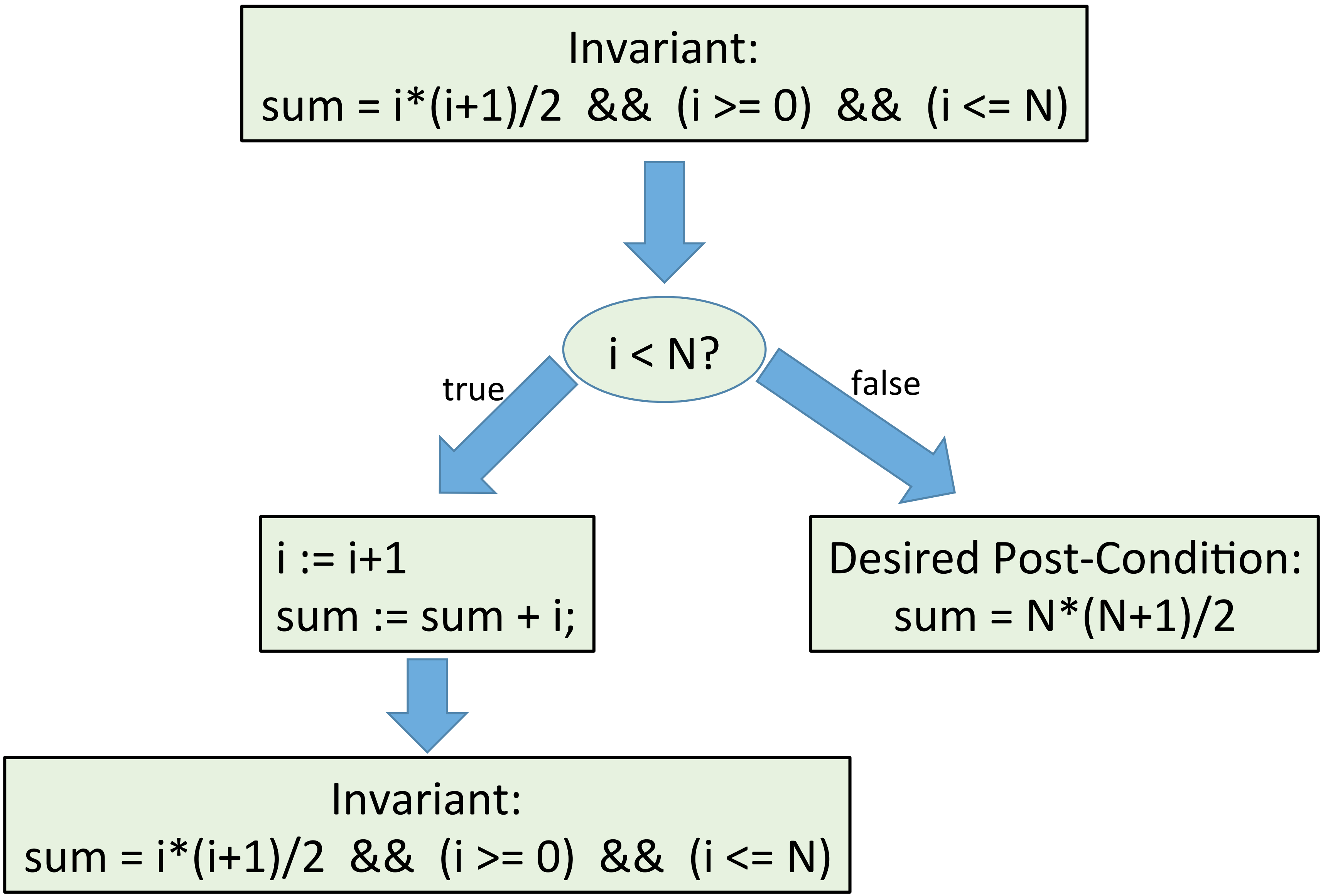
$$\vdash \{P \wedge E\}\ S{\downarrow}1 \ \{Q\}, \quad \vdash \{P \wedge \neg E\}\ S{\downarrow}2 \ \{Q\} / \vdash \{P\}\ \text{if } (E) \text{ then } S{\downarrow}1 \ \text{ else } S{\downarrow}2 \ \{Q\}$$

Iteration

$$? / \vdash \{P\} \text{ while } (E) \text{ do } S \{Q\}$$

Demo

```
method Sum (N: int)
    returns (sum : int)
    requires N > 0;
    ensures sum == N*(N+1)/2;
{
    var i := 0; sum := 0;
    while (i < N)
       invariant (sum == i*(i+1)/2)
          && (i >= 0) && (i <= N)
    {
        i := i + 1;
        sum := sum + i;
    }
}
```


Invariant:
sum = i*(i+1)/2 && (i >= 0) && (i <= N)
i < N?
true
false
i := i+1
sum := sum + i;
Desired Post-Condition:
sum = N*(N+1)/2
Invariant:
sum = i*(i+1)/2 && (i >= 0) && (i <= N)

Iteration

$$?/\vdash \{P\}\ \text{while}\ (E)\ \text{do}\ S\ \{Q\}$$

$$\vdash \{P \wedge E\}\ S\ \{P\}, \quad \vdash (P \wedge \neg E) \Rightarrow Q / \vdash \{P\}\ \text{while}\ (E)\ \text{do}\ S\ \{Q\}$$

Assignment

$$? / \vdash \{P\}\, x := E\, \{Q\}$$

$$/ \vdash \{Q[x \to E]\}\, x := E\, \{Q\}$$

Exercise

- Try out the assignment rule

$\{2*x > 10\}$

- $\{?\}\ \ y := 2*x\ \{\, y > 10\,\}$

- $\{?\}\ x := x+1\ \{\, x > 10\,\}$

$\{x+1 > 10\}$

Precondition Strengthening

$$\frac{\vdash \{R\}\, S\, \{Q\}, \quad P \Rightarrow R}{\vdash \{P\}\, S\, \{Q\}}$$

- (Dijsktra)
- Weakest liberal precondition (wlp)
- $wlp(S,Q)$ is defined to be the weakest condition P such that $\{P\}\, S\, \{Q\}$ holds.

Postcondition Weakening

$$\vdash \{P\}\, S\, \{R\}, \quad R \Rightarrow Q \,/\, \vdash \{P\}\, S\, \{Q\}$$

- Strongest postcondition (sp)
- $sp(S,P)$ is defined to be the strongest condition Q such that $\{P\}\, S\, \{Q\}$ holds

Soundness of Hoare Logic

$$\vdash \{P\}\ S\ \{Q\}$$

$$\Downarrow$$

$$\sigma \in [\![P]\!] \wedge \sigma \leadsto_S^{*} \sigma' \Rightarrow \sigma' \in [\![Q]\!]$$

Concurrency

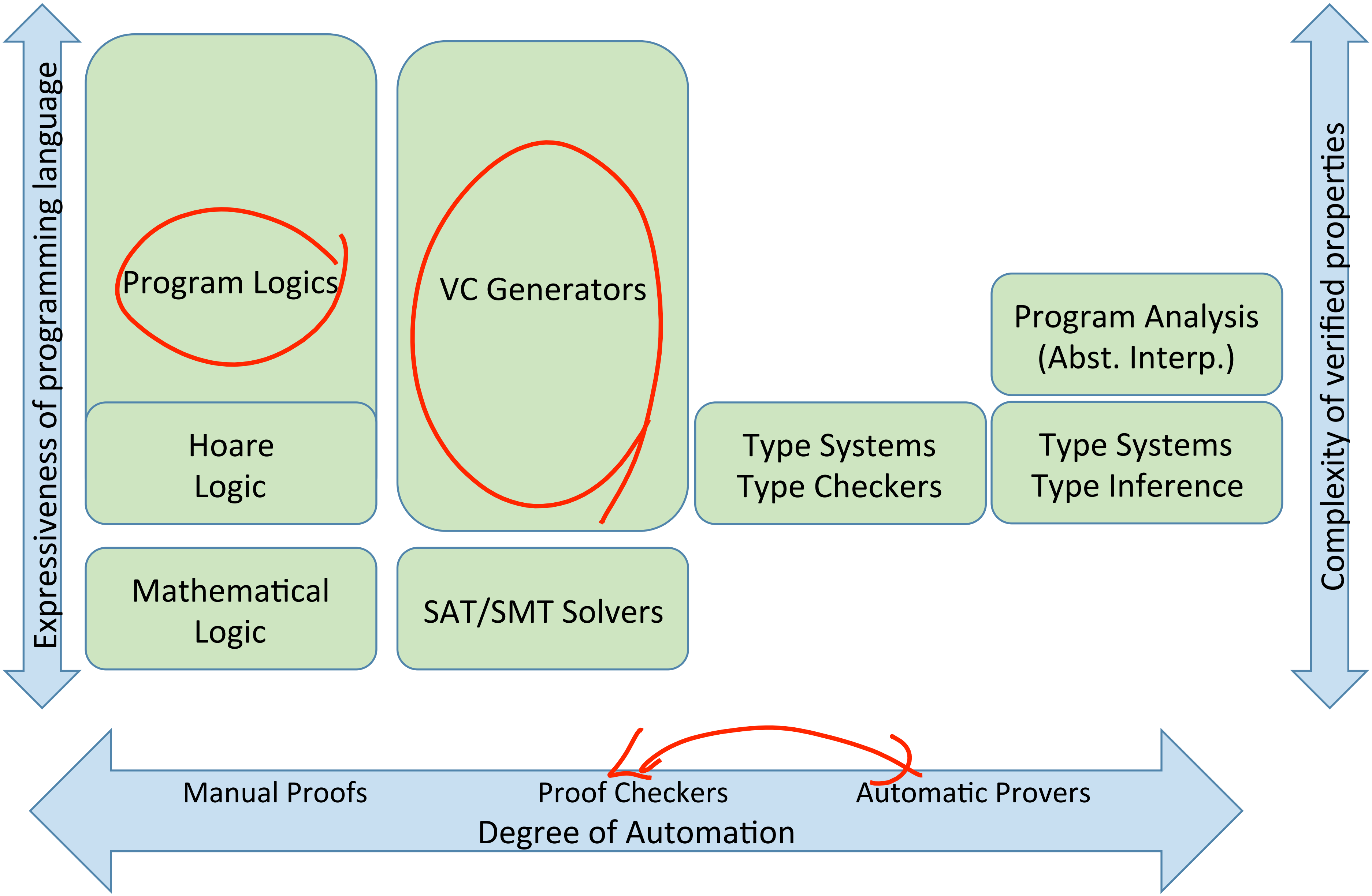
- Separation Logic

Inductive Invariants

```
method Sum (N: int)
    returns (sum : int)
    requires N > 0;
    ensures sum == N*(N+1)/2;
{
    var i := 0;
    while (i < N)
        invariant sum == i*(i+1)/2
   {
        i := i + 1;
        sum := sum + i;
    }
}
```

Inductive Invariants

- Inferring inductive invariants when users do not specify them
  - Hard
  - Program analysis (abstract interpretation)
- Inferring inductive invariants with some user hints/guidance


Expressiveness of programming language
Program Logics
VC Generators
Program Analysis
(Abst. Interp.)
Complexity of verified properties
Hoare
Logic
Type Systems
Type Checkers
Type Systems
Type Inference
Mathematical
Logic
SAT/SMT Solvers
Manual Proofs
Proof Checkers
Automatic Provers
Degree of Automation