Syntax-Guided Synthesis

Rajeev Alur

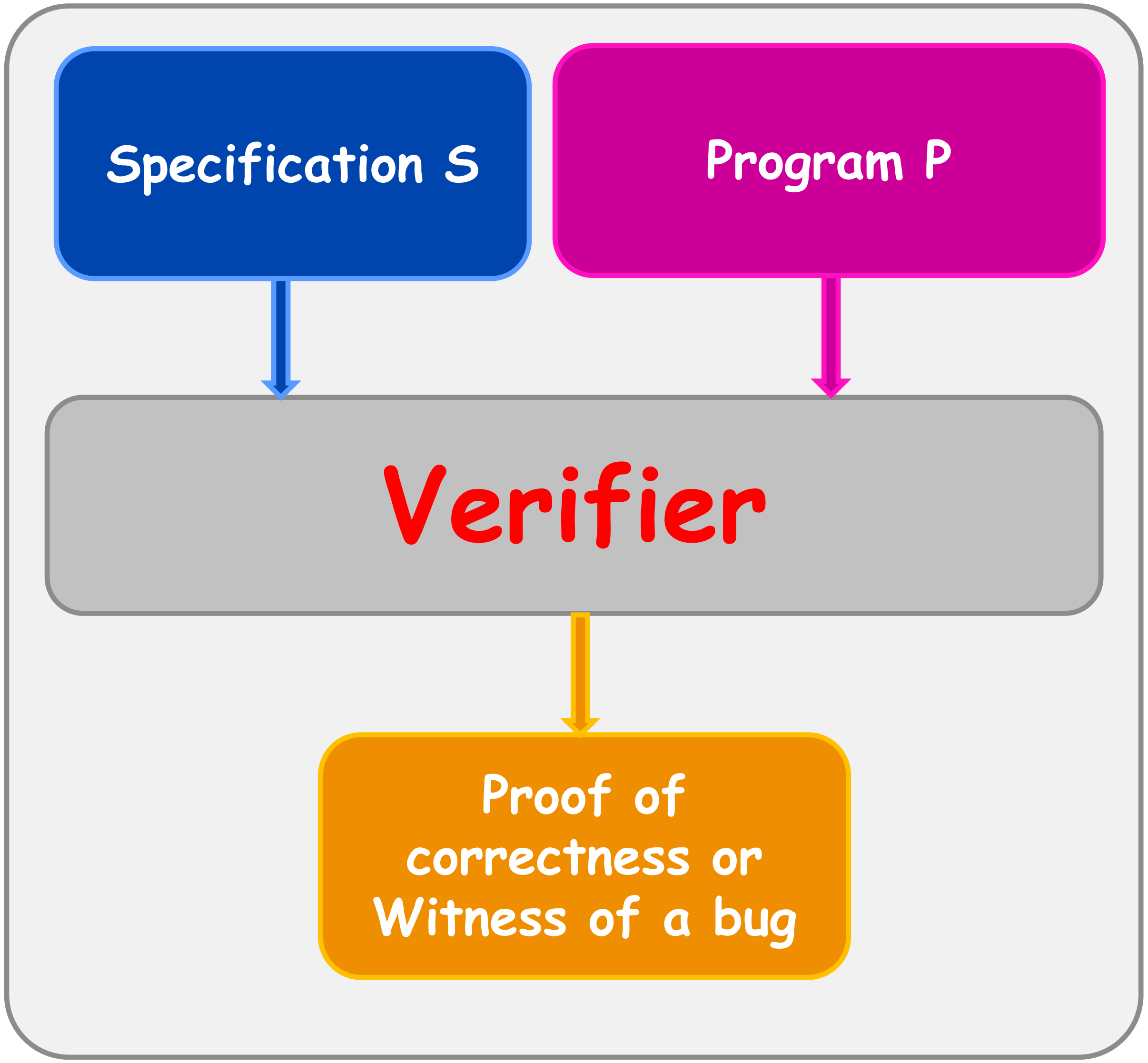
Marktoberdorf Summer School 2014

ExCAPE
Expeditions in Computer Augmented Program Engineering

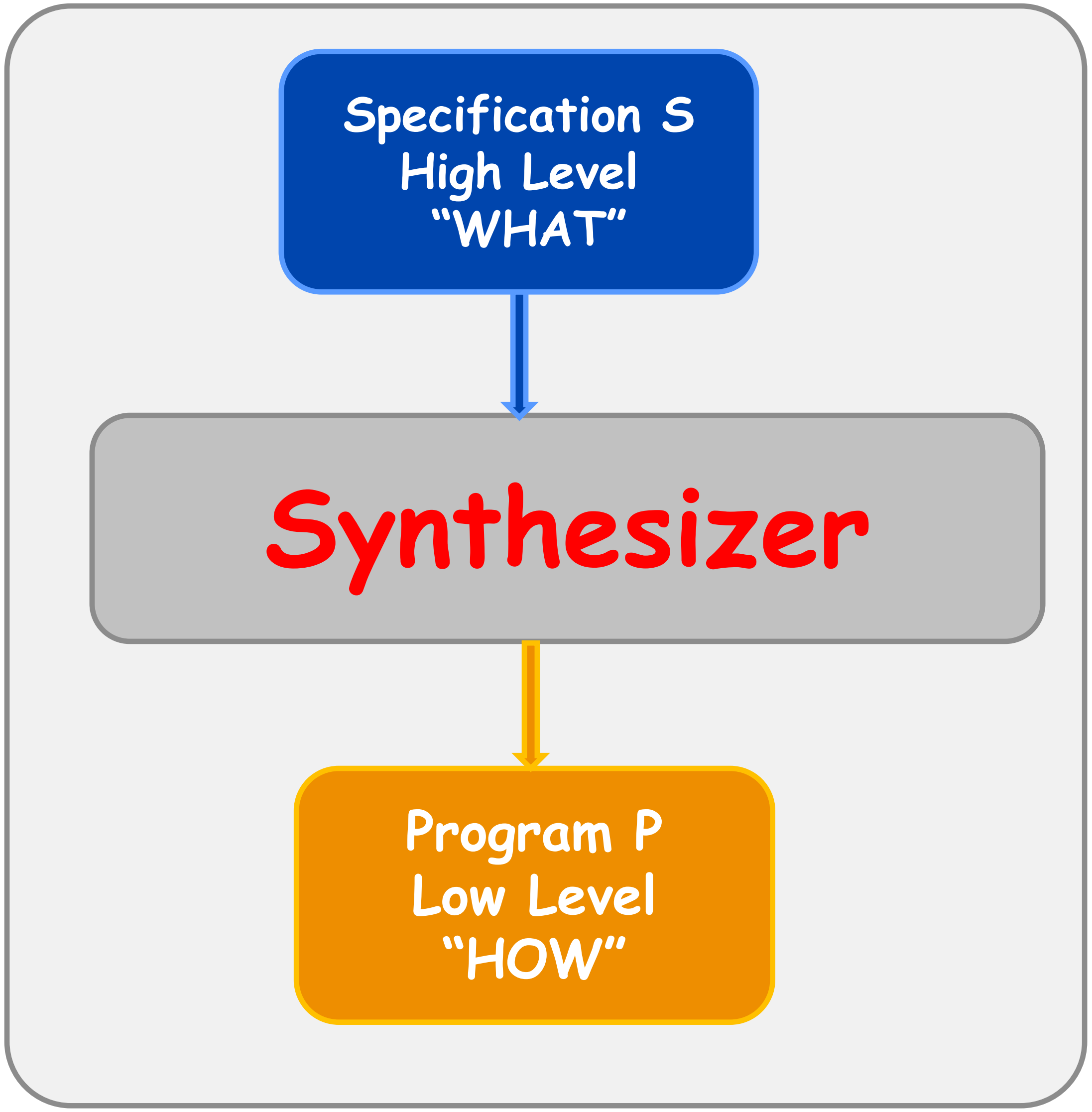
SyGuS

Penn Engineering

Program Verification

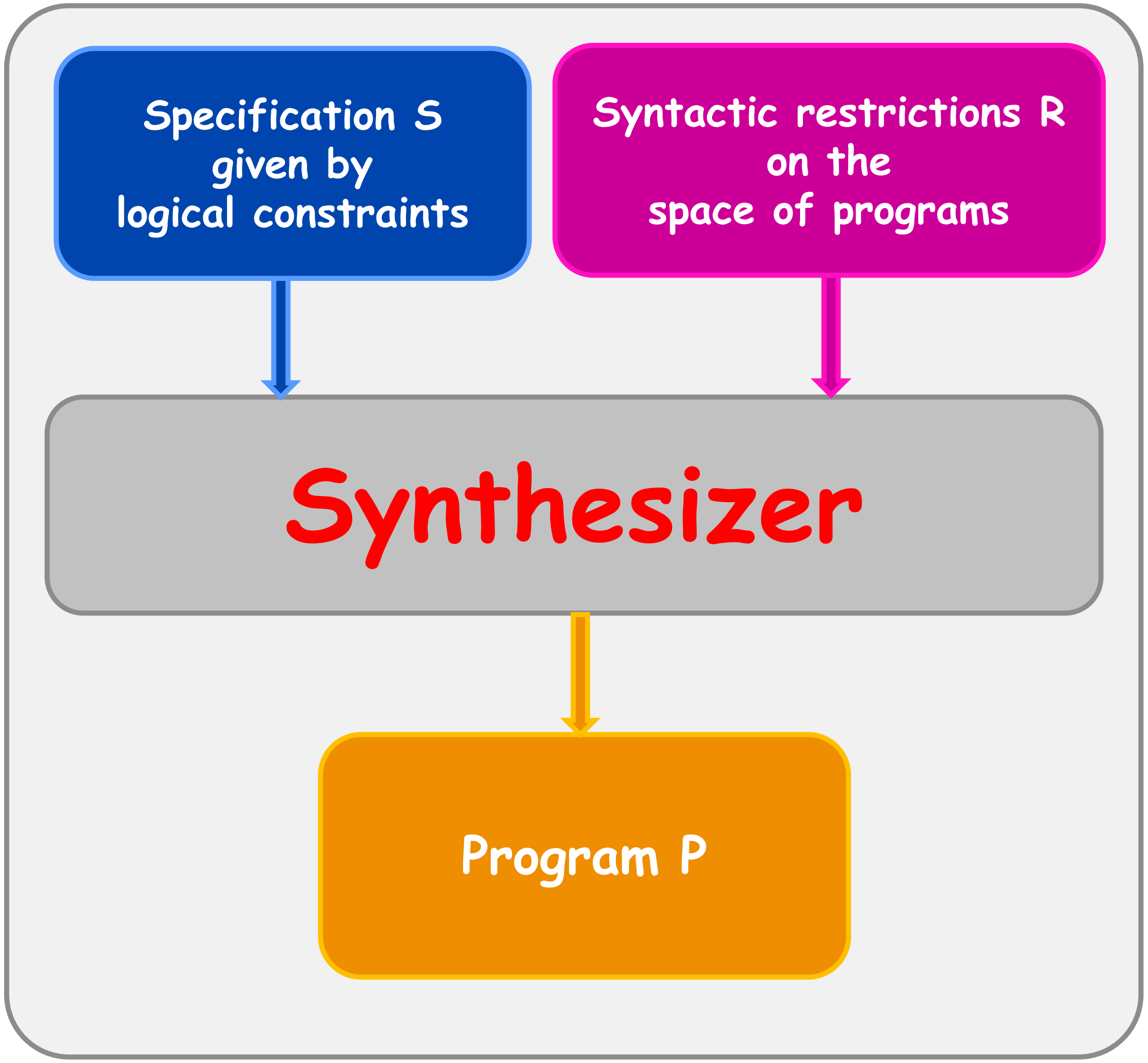


Classical Program Synthesis



Syntax-Guided Synthesis

www.sygus.org



Specification S given by logical constraints
Syntactic restrictions R on the space of programs
Synthesizer
Program P

References

- Syntax Guided Synthesis

  R. Alur, R. Bodik, G. Juniwal, M. Martin, M. Raghothaman, S.Seshia, R. Singh, A. Solar-Lezama, E. Torlak, A. Udupa

  Proc. FMCAD, 2013

- TRANSIT: Specifying protocols with concolic snippets

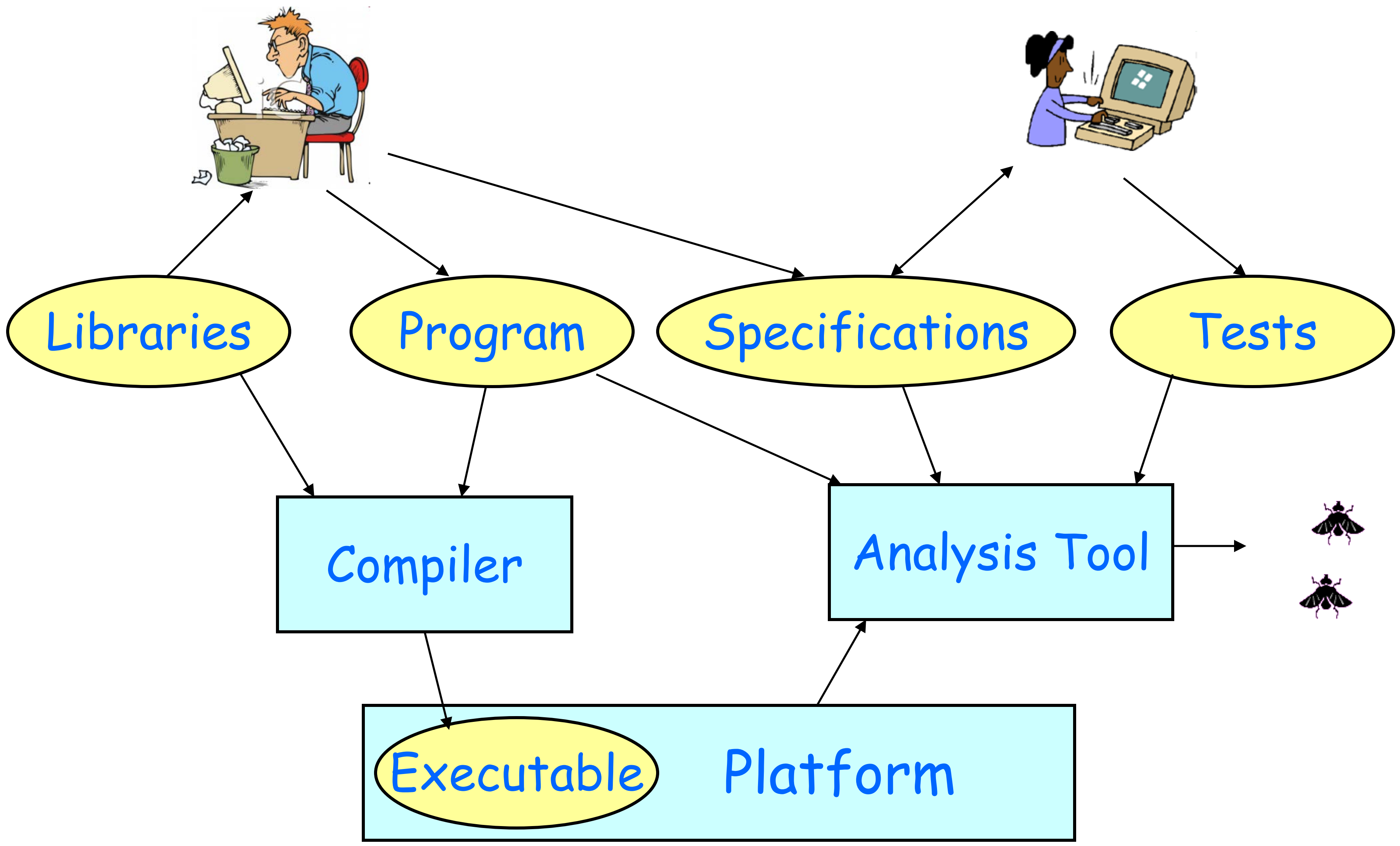
  A. Udupa, A. Raghavan, J. Deshmukh, S. Mador-Haim, M. Martin, R. Alur

  Proc. PLDI 2013

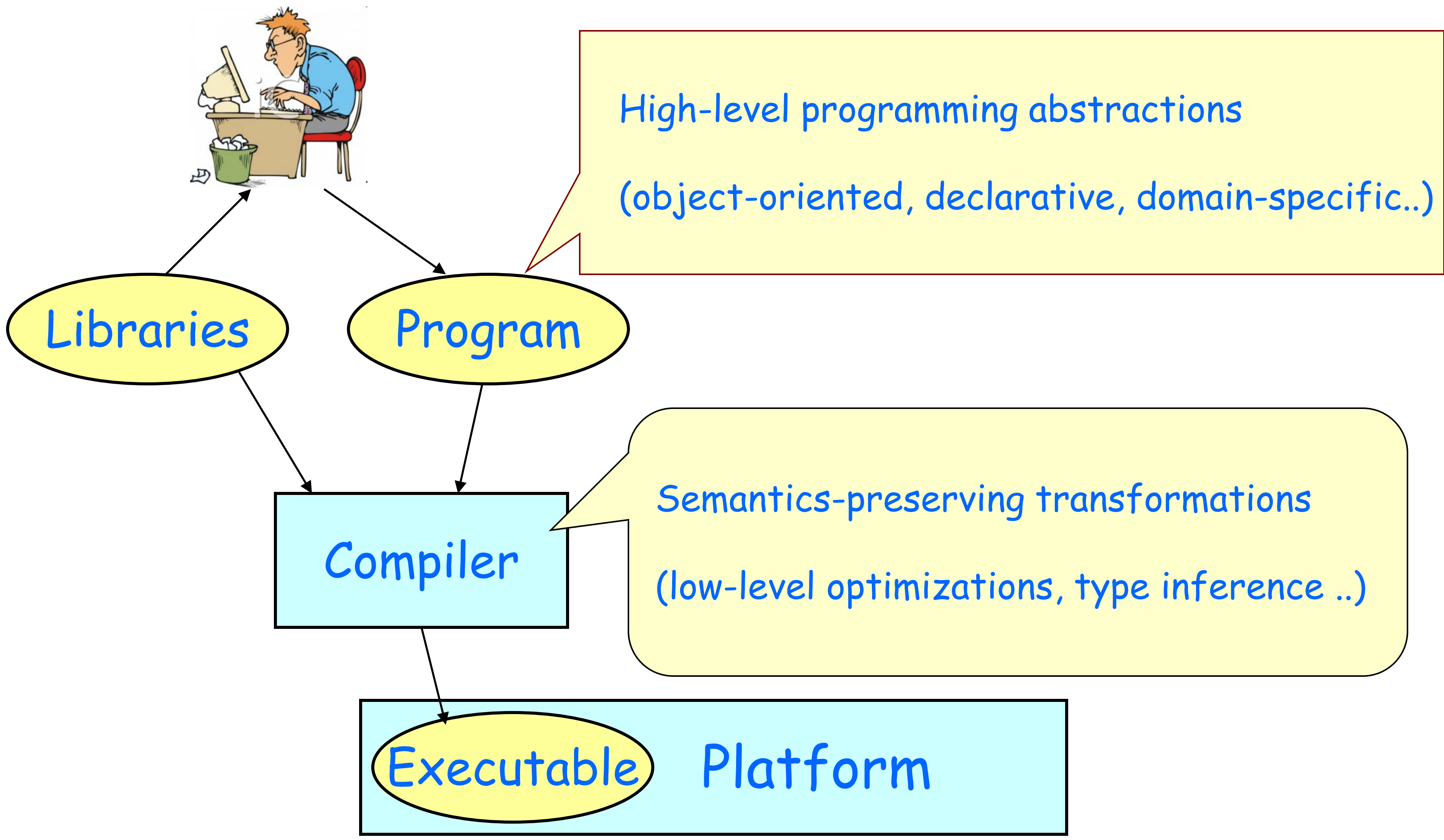
Outline of Lectures

- ❖ Program Verification and SMT Solvers
- ❑ Motivation for Syntax-Guided Synthesis (SyGuS)
- ❑ Formalization of SyGuS
- ❑ Solving SyGuS
- ❑ TRANSIT for Protocol Specification

Software Design



Programming Technology



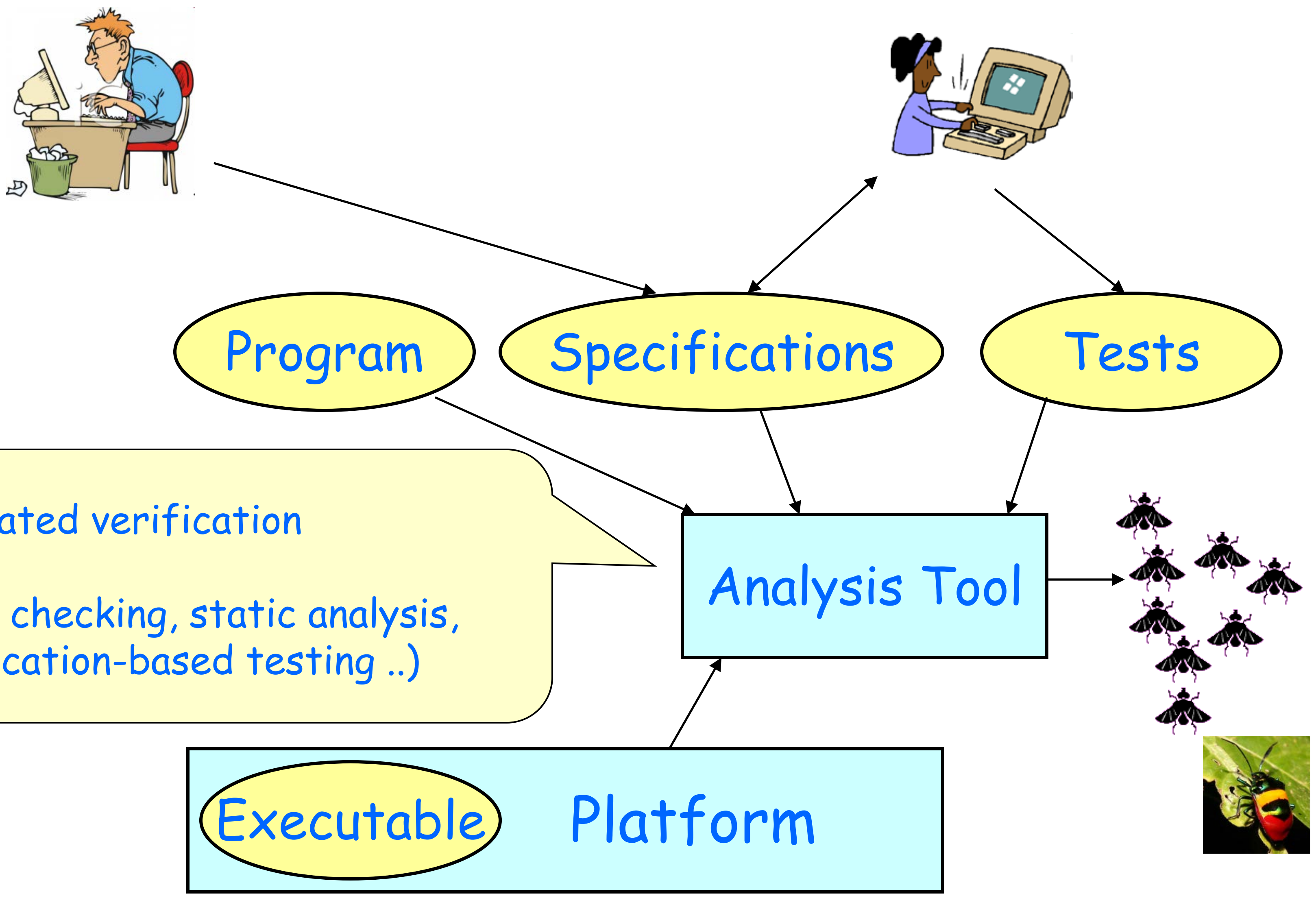
High-level programming abstractions

(object-oriented, declarative, domain-specific..)

Semantics-preserving transformations

(low-level optimizations, type inference ..)

Verification Technology



Automated verification

(model checking, static analysis, specification-based testing ..)

Program Verification

- Does a program P meet its specification φ ?
- Historical roots: Hoare logic for formalizing correctness of structured programs (late 1960s)
- Early examples: sorting, graph algorithms
- Provides calculus for pre/post conditions of structured programs

Sample Proof: Selection Sort

```
SelectionSort(int A[],n) {
  i := 0;
  while(i < n-1) {
    v := i;
    j := i + 1;
    while (j < n) {
      if (A[j]<A[v])
        v := j ;
      j++;
    }
    swap(A[i], A[v]);
    i++;
  }
  return A;
}
```

Invariant:
$\forall k1,k2.\ (0 \le k1 < k2 < n) \land (k1 < i)$
$\Rightarrow A[k1] \le A[k2]$

Invariant:
$i < j \land$
$i \le v < n \land$
$(\forall k1,k2.\ (0 \le k1 < k2 < n) \land (k1 < i)$
$\Rightarrow A[k1] \le A[k2]) \land$
$(\forall k.\ (i \le k < j) \land (k \ge 0)$
$\Rightarrow A[v] \le A[k])$

post: $\forall k : 0 \le k < n \Rightarrow A[k] \le A[k + 1]$

Towards Practical Program Verification

1. Focus on simpler verification tasks:
   - Not full functional correctness, just absence of specific errors
   - Success story: Array accesses are within bounds

2. Provide automation as much as possible
   - Program verification is undecidable
   - Programmer asked to give annotations when absolutely needed
   - Consistency of annotations checked by SMT solvers

3. Use verification technology for synergistic tasks
   - Directed testing
   - Bug localization

Selection Sort: Array Access Correctness

```
SelectionSort(int A[],n) {
  i := 0;
  while(i < n-1) {
    v := i;
    j := i + 1;
    while (j < n) {
      assert (0 ≤ j < n) & (0 ≤ v < n)
      if (A[j]<A[v])
        v := j ;
      j++;
    }
    assert (0 ≤ i <n) & (0 ≤ v < n)
    swap(A[i], A[v]);
    j++;
  }
  return A;
}
```

Selection Sort: Proving Assertions

```
SelectionSort(int A[],n) {
  i := 0;
  while(i < n-1) {
    v := i;
    j := i + 1;
    while (j < n) {
      assert 0≤j<n & 0≤v<n
      if (A[j]<A[v])
        v := j ;
      j++;
    }
    assert (0 ≤ i < n) & 0 ≤ v<n
    swap(A[i], A[v]);
    i++;
  }
  return A;
}
```

Check validity of formula

$(i = 0) \,\&\, (i < n-1) \Rightarrow (0 \le i < n)$

And validity of formula

$(0 \le i < n) \,\&\, (i' = i+1) \,\&\, (i' < n-1)$
$\Rightarrow (0 \le i' < n)$

Discharging Verification Conditions

- Check validity of

  $(i = 0)\ \&\ (i < n-1) \Rightarrow (0 \le i < n)$

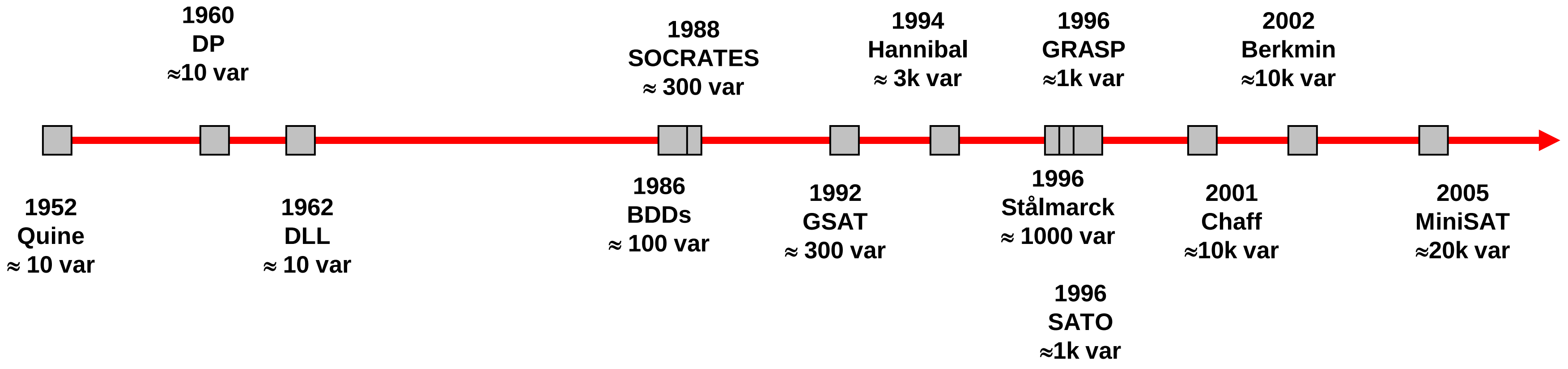
- Reduces to checking satisfiability of

  $(i = 0)\ \&\ (i < n-1)\ \&\ \sim(0 \le i < n)$

- Core computational problem: checking satisfiability
  - Classical satisfiability: SAT
    Boolean variables + Logical connectives
  - SMT: Constraints over typed variables
    i and n are of type Integer or BitVector[32]

A Brief History of SAT

- Fundamental Thm of CS: SAT is NP-complete (Cook, 1971)
  - Canonical computationally intractable problem
  - Driver for theoretical understanding of complexity
- Enormous progress in scale of problems that can be solved
  - Inference: Discover new constraints dynamically
  - Exhaustive search with pruning
  - Algorithm engineering: Exploit architecture for speed-up
- SAT solvers as the canonical computational hammer!



SMT: Satisfiability Modulo Theories

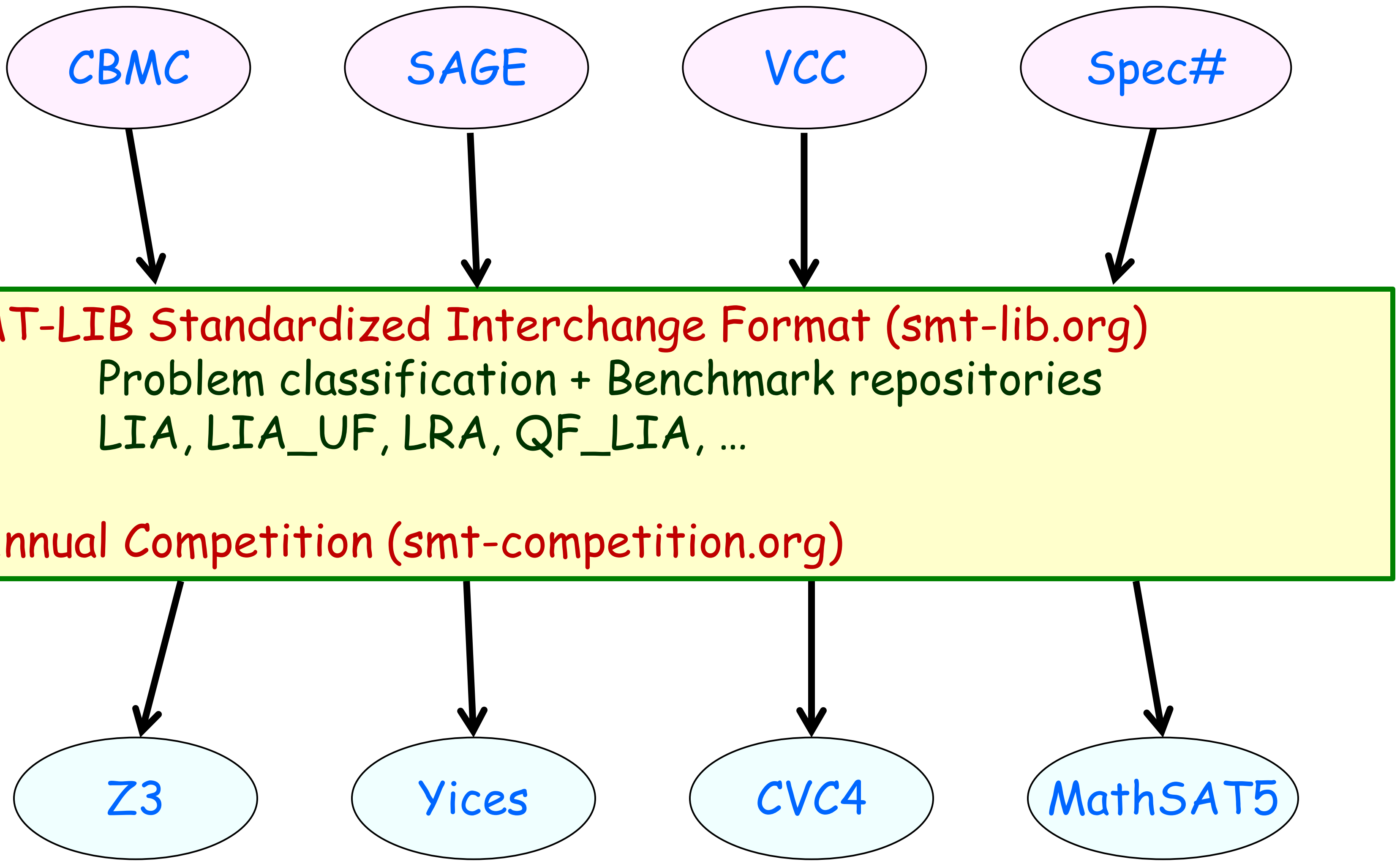
- Computational problem: Find a satisfying assignment to a formula
  - Boolean + Int types, logical connectives, arithmetic operators
  - Bit-vectors + bit-manipulation operations in C
  - Boolean + Int types, logical/arithmetic ops + Uninterpreted functs
- "Modulo Theory": Interpretation for symbols is fixed
  - Can use specialized algorithms (e.g. for arithmetic constraints)
- Progress in improved SMT solvers

Little Engines of Proof

SAT; Linear arithmetic; Congruence closure

SMT Success Story

SMT Solvers ⟷ Verification Tools



SMT-LIB Standardized Interchange Format (smt-lib.org)

Problem classification + Benchmark repositories

LIA, LIA_UF, LRA, QF_LIA, …

+ Annual Competition (smt-competition.org)

Synthesis Puzzle 1: Prisoners and a switch

There are N prisoners who get together initially to decide on a strategy. Then, each prisoner is taken to own isolated cell. A guard goes to a cell and brings its occupant to a room with a switch. The switch can be either up or down. The prisoner can inspect the switch, then can decide to leave it as it is or flick it, and is then taken back to the cell. The guard repeats this process infinitely often. The order in which the prisoners are brought to the cell is arbitrary. However, the guard assures fairness: every prisoner will visit the room infinitely often. At any time, a prisoner can declare "I know for sure that every prisoner has visited the room with the switch at least once." When the guard hears this declaration, if the statement is indeed correct, all prisoners are set free, but if the statement is false, all prisoners are destined to stay imprisoned forever.

What strategy should the prisoners use to ensure their eventual freedom?

Reference: Cartalk Puzzler (see also Rustan Leino's page of puzzles)

Outline of Lectures

✓ Program Verification and SMT Solvers

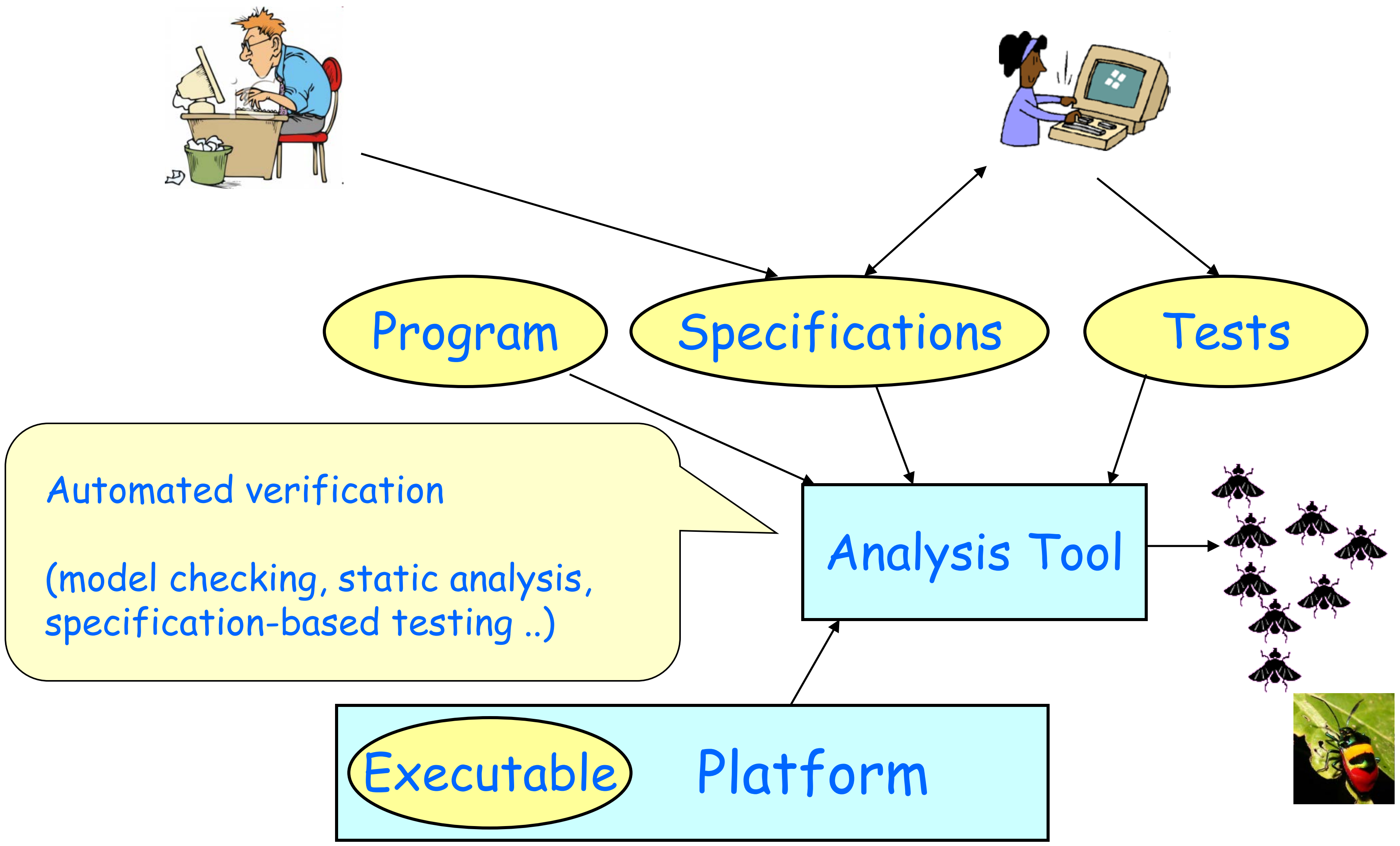
❖ Motivation for Syntax-Guided Synthesis (SyGuS)

❑ Formalization of SyGuS

❑ Solving SyGuS

❑ TRANSIT for Protocol Specification

Verification Technology



Challenges

- Software development still remains expensive and error-prone…
- What it means to “code” hasn’t changed…
- Verification/testing done after design
    - Costly system design cycle
    - Many reported bugs not fixed
- Computing power is transforming many engineering disciplines with the notable exception of programming itself

Opportunities

- Enormous computing power available on desktops of today's programmers
- Impressive strides in formal verification technology
  - Highly optimized SAT solvers that can solve real-world problems
  - Off-the-shelf tools for static analysis, machine learning…
- Demand for new software development approaches
  - Receptive industry
  - Shifting goal of system design from performance to predictability

Synthesis: A Plausible Solution?

- Classical: Mapping a high-level (e.g. logical) specification to an executable implementation

- Benefits of synthesis:
  - Make programming easier: Specify "what" and not "how"
  - Eliminate costly gap between programming and verification

- Impressive progress, but …
  - High computational complexity
  - Writing complete logical specifications is a challenging task

- Recent shift in focus: simpler synthesis tasks

Parallel Parking by Sketching

Ref: Chaudhuri, Solar-Lezama (PLDI 2010)

```
Err = 0.0;
for(t = 0; t<T; t+=dT){
  if(stage==STRAIGHT){
    if(t > ??) stage= INTURN;
  }
  if(stage==INTURN){
    car.ang = car.ang - ??;
    if(t > ??) stage= OUTTURN;
  }
  if(stage==OUTTURN){
    car.ang = car.ang + ??;
    if(t > ??) break;
  }
  simulate_car(car);
  Err += check_collision(car);
}
Err += check_destination(car);
```

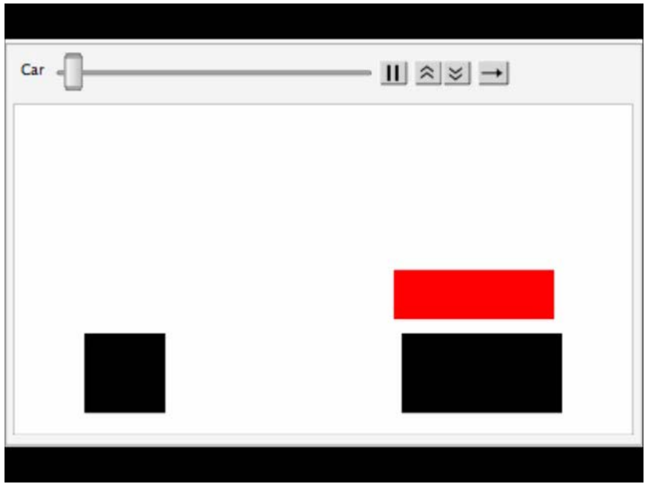
When to start turning?

Backup straight

How much to turn?

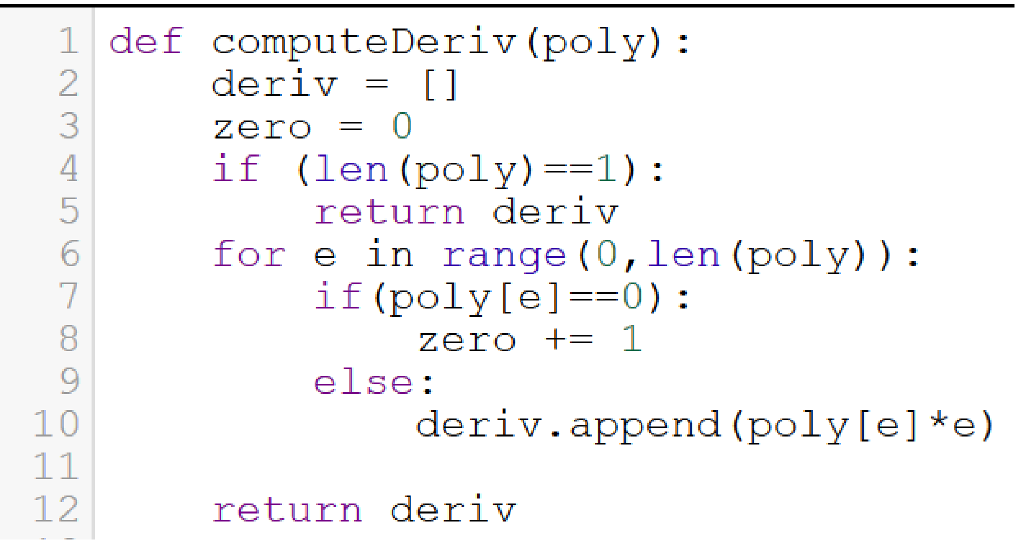
Turn

Straighten



Autograder: Feedback on Programming Homeworks

Singh et al (PLDI 2013)



```
 1  def computeDeriv(poly):
 2      deriv = []
 3      zero = 0
 4      if (len(poly)==1):
 5          return deriv
 6      for e in range(0,len(poly)):
 7          if(poly[e]==0):
 8              zero += 1
 9          else:
10              deriv.append(poly[e]*e)
11
12      return deriv
```

Student Solution P
+ Reference Solution R
+ Error Model

The program requires **3** changes:

- In the return statement **return deriv** in **line 5**, replace **deriv** by **[0]**.
- In the comparison expression **(poly[e] == 0)** in **line 7**, change **(poly[e] == 0)** to **False**.
- In the expression **range(0, len(poly))** in **line 6**, replace **0** by **1**.

Find min no of edits to P so as to make it equivalent to R

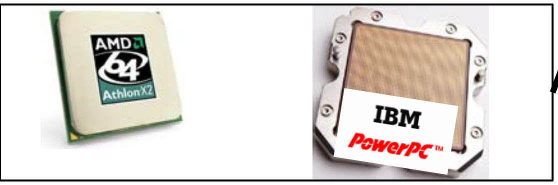
Paraglide: From Sequential to Parallel Code

Ref: Vechev et al (POPL 2010)

Sequential Program

```
bool add(int key){
 atomic
   Entry *pred,*curr,*entry
   locate(pred,curr,key);
   k = (curr->key == key)
   if (k) return false
   entry = new Entry()
   entry->next = curr
   pred->next = entry
   return true
}
```

Architecture Description



Paraglide

Minimal Synchronization

```
bool add(int key) {
 Entry *pred,*curr,*entry
restart:
 locate(pred,curr,key)
 k = (curr->key == key)
 if (k) return false
 entry = new Entry()
 entry->next = curr
 val= CAS(&pred->next,<curr,0>,<entry,0>)
 if (!val) goto restart
 return true
}
```

✓

- Target: Highly concurrent work queue in C/C++
- Infers minimal number of fences needed for synchronization
- Unexpected, correct, minimal solutions now deployed in IBM

FlashFill: Programming by Examples

Ref: Gulwani (POPL 2011)

Input	Output
(425)-706-7709	425-706-7709
510.220.5586	510-220-5586
1 425 235 7654	425-235-7654
425 745-8139	425-745-8139

- Infers desired Excel macro program
- Iterative: user gives examples and corrections
- Incorporated in commercial version of Microsoft Excel

Superoptimizing Compiler

- Given a program P, find a "better" equivalent program P'

```
 multiply (x[1,n], y[1,n]) {
   x1 = x[1,n/2];
   x2 = x[n/2+1, n];
   y1 = y[1, n/2];
   y2 = y[n/2+1, n];
   a = x1 * y1;
   b = shift( x1 * y2, n/2);
   c = shift( x2 * y1, n/2);
   d = shift( x2 * y2, n);
   return ( a + b + c + d)
}
```

Replace with equivalent code with only 3 multiplications

Automatic Invariant Generation

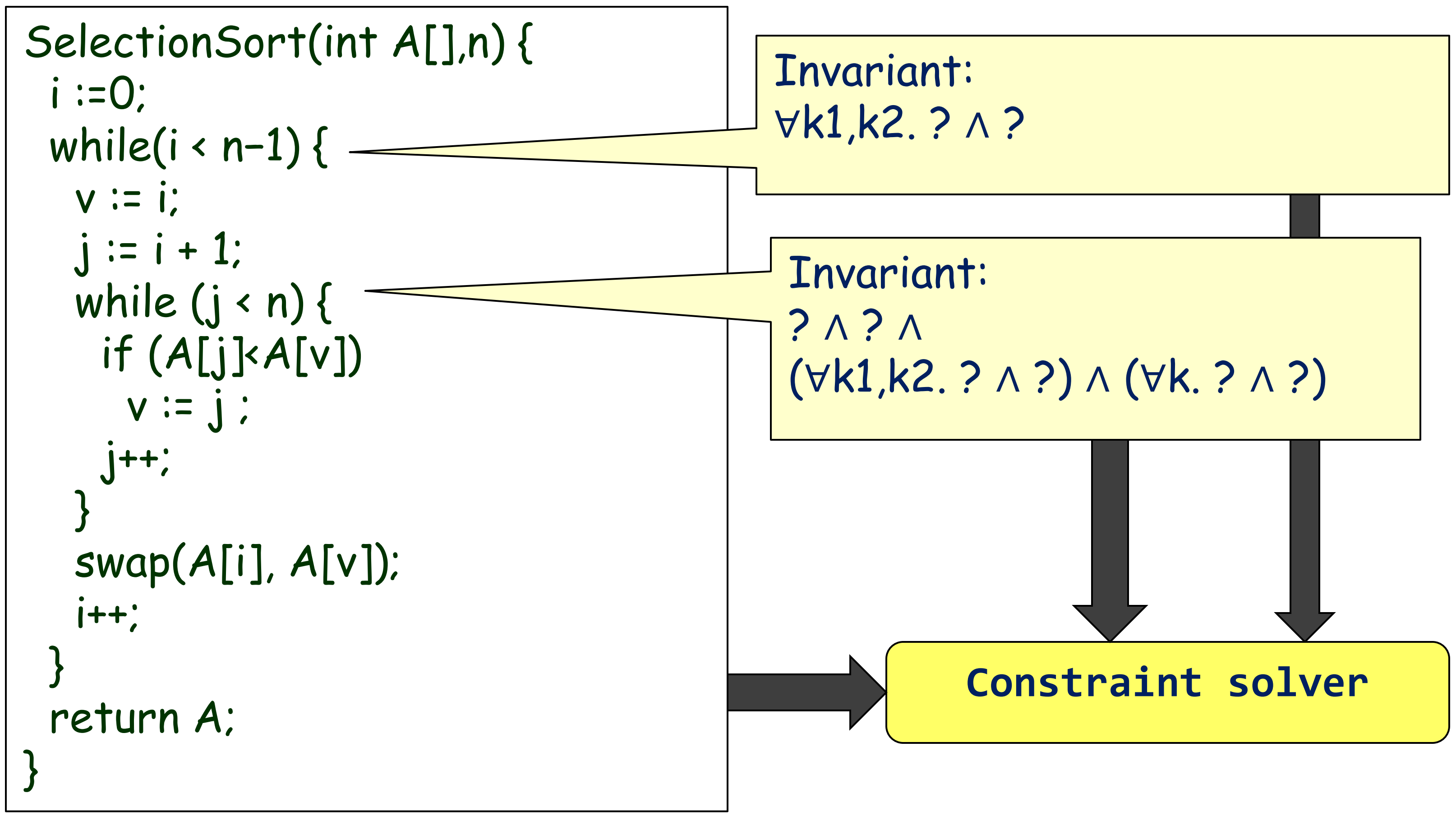
```
SelectionSort(int A[],n) {
  i := 0;
  while(i < n−1) {
    v := i;
    j := i + 1;
    while (j < n) {
      if (A[j]<A[v])
        v := j ;
      j++;
    }
    swap(A[i], A[v]);
    i++;
  }
  return A;
}
```

Invariant: ?

Invariant: ?

post: $\forall k : 0 \le k < n \Rightarrow A[k] \le A[k+1]$

Template-based Automatic Invariant Generation



```
SelectionSort(int A[],n) {
  i :=0;
  while(i < n−1) {
    v := i;
    j := i + 1;
    while (j < n) {
      if (A[j]<A[v])
        v := j ;
      j++;
    }
    swap(A[i], A[v]);
    i++;
  }
  return A;
}
```

post: $\forall k : 0 \le k < n \Rightarrow A[k] \le A[k+1]$

Template-based Automatic Invariant Generation

```
SelectionSort(int A[],n) {
  i :=0;
  while(i < n−1) {
    v := i;
    j := i + 1;
    while (j < n) {
      if (A[j]<A[v])
        v := j ;
      j++;
    }
    swap(A[i], A[v]);
    i++;
  }
  return A;
}
```

Invariant:
$\forall k1,k2.\ 0 \le k1 < k2 < n \wedge$
$k1 < i \Rightarrow A[k1] \le A[k2]$

Invariant:
$i < j \wedge$
$i \le v < n \wedge$
$(\forall k1,k2.\ 0 \le k1 < k2 < n \wedge$
$k1 < i \Rightarrow A[k1] \le A[k2]) \wedge$
$(\forall k.\ i1 \le k < j \wedge$
$k \ge 0 \Rightarrow A[v] \le A[k])$

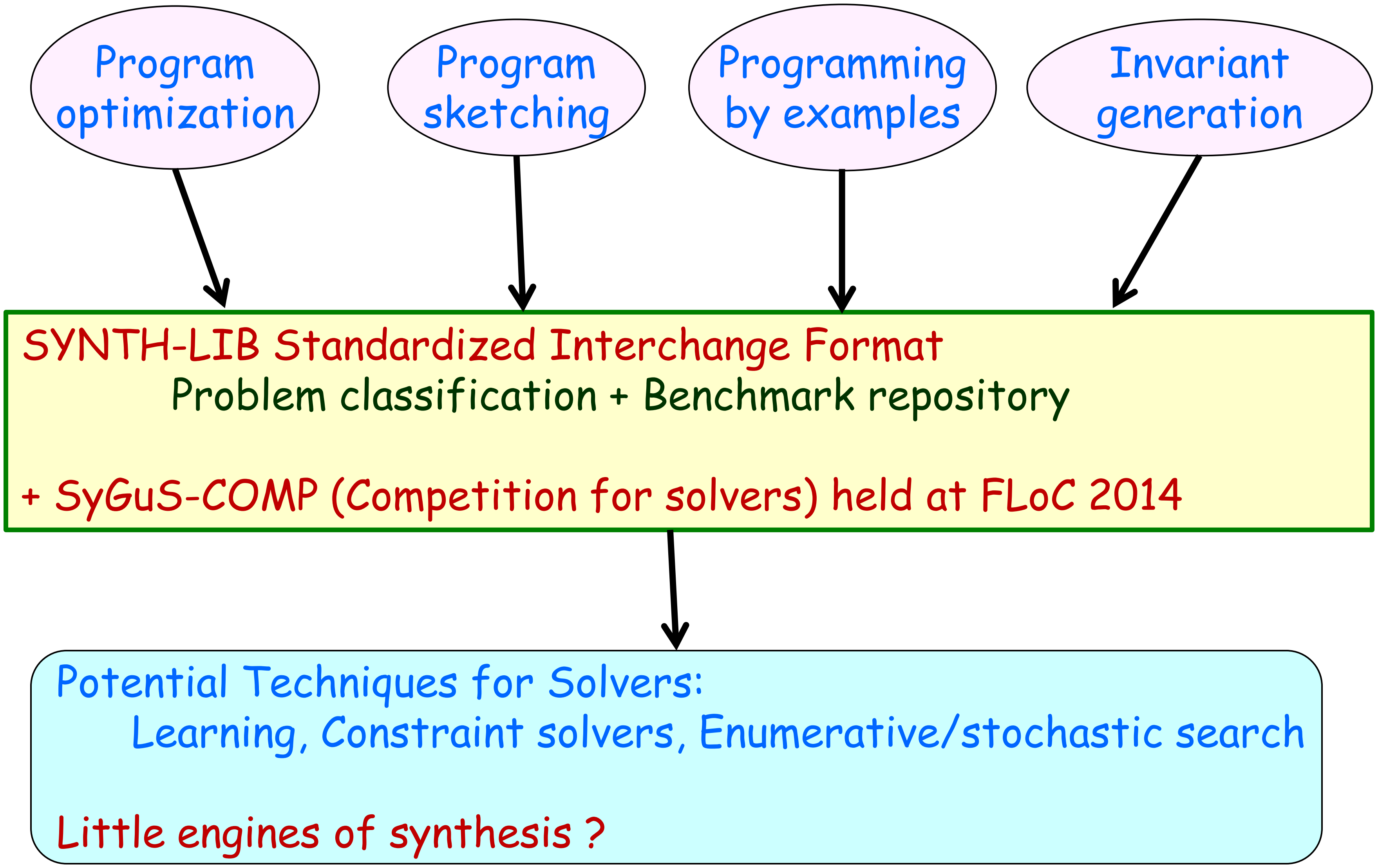
post: $\forall k : 0 \le k < n \Rightarrow A[k] \le A[k + 1]$

Syntax-Guided Program Synthesis

- Core computational problem: Find a program P such that
  1. P is in a set E of programs (syntactic constraint)
  2. P satisfies spec φ (semantic constraint)

- Common theme to many recent efforts
  - Sketch (Bodik, Solar-Lezama et al)
  - FlashFill (Gulwani et al)
  - Super-optimization (Schkufza et al)
  - Invariant generation (Many recent efforts…)
  - Genetic programming + model checking (Peled et al)
  - TRANSIT for protocol synthesis (Udupa et al)
  - Oracle-guided program synthesis (Jha et al)
  - Implicit programming: Scala^Z3 (Kuncak et al)
  - Auto-grader (Singh et al)

SyGuS Solvers $\longleftrightarrow$ Synthesis Tools



SYNTH-LIB Standardized Interchange Format

Problem classification + Benchmark repository

+ SyGuS-COMP (Competition for solvers) held at FLoC 2014

Potential Techniques for Solvers:

Learning, Constraint solvers, Enumerative/stochastic search

Little engines of synthesis ?

Prisoners and a switch: Solution

Suppose the switch is initially ON, and all know this fact.

Initially prisoners elect a leader.

Strategy for non-leader:

- If the switch is OFF, leave it as it is,
- else turn it OFF, provided you have never done this before.

Strategy for leader:

- Leader maintains a counter, initially 0
- If the switch is ON, leave it as it is,
- else turn it ON and increment counter.
- If counter equals N-1, declare "everyone has visited at least once"

Exercise: what if the initial state of switch is not known?

Outline of Lectures

- ✓ Program Verification and SMT Solvers
- ✓ Motivation for Syntax-Guided Synthesis (SyGuS)
- ❖ Formalization of SyGuS
- ❑ Solving SyGuS
- ❑ TRANSIT for Protocol Specification

Syntax-Guided Synthesis (SyGuS) Problem

- Fix a background theory T: fixes types and operations

- Function to be synthesized: name f along with its type
  - General case: multiple functions to be synthesized

- Inputs to SyGuS problem:
  - Specification φ
    Typed formula using symbols in T + symbol f
  - Set E of expressions given by a context-free grammar
    Set of candidate expressions that use symbols in T

- Computational problem:
  Output e in E such that φ[f/e] is valid (in theory T)

SyGuS Example

- Theory QF-LIA (Quantifier-free linear integer arithmetic)
  - Types: Integers and Booleans
  - Logical connectives, Conditionals, and Linear arithmetic
  - Quantifier-free formulas

- Function to be synthesized f (int x, int y) : int

- Specification: $(x \leq f(x,y))$ & $(y \leq f(x,y))$ & $(f(x,y) = x \mid f(x,y) = y)$

- Candidate Implementations: Linear expressions
  - LinExp := x | y | Const | LinExp + LinExp | LinExp - LinExp

- No solution exists

SyGuS Example

- Theory QF-LIA
- Function to be synthesized: f (int x, int y) : int
- Specification: $(x \le f(x,y))$ & $(y \le f(x,y))$ & $(f(x,y) = x \mid f(x,y)=y)$
- Candidate Implementations: Conditional expressions without +

  Term := x | y | Const | If-Then-Else (Cond, Term, Term)
  Cond := Term <= Term | Cond & Cond | ~ Cond | (Cond)

- Possible solution:

  If-Then-Else ($x \le y$, y, x)

Let Expressions and Auxiliary Variables

- Synthesized expression maps directly to a straight-line program
- Grammar derivations correspond to expression parse-trees
- How to capture common subexpressions (which map to aux vars) ?
- Solution: Allow "let" expressions
- Candidate-expressions for a function f(int x, int y) : int

```
T := (let [z = U] in  z + z)
U := x | y | Const | (U) | U + U | U*U
```

Optimality

- Specification for f(int x) : int
  - $x \le f(x)$ & $-x \le f(x)$
- Set E of implementations: Conditional linear expressions
- Multiple solutions are possible
  - If-Then-Else ($0 \le x$, x, 0)
  - If-Then-Else ($0 \le x$, x, -x)
- Which solution should we prefer?
  - Need a way to rank solutions (e.g. size of parse tree)

From SMT-LIB to SYNTH-LIB

```
(set-logic LIA)
(synth-fun max2 ((x Int) (y Int)) Int
    ((Start Int (x y 0 1
                (+ Start Start)
                (- Start Start)
                (ite StartBool Start Start)))
     (StartBool Bool ((and StartBool StartBool)
                      (or StartBool StartBool)
                      (not StartBool)
                      (<= Start Start))))
 (declare-var x Int)
 (declare-var y Int)
 (constraint (>= (max2 x y) x))
 (constraint (>= (max2 x y) y))
 (constraint (or (= x (max2 x y)) (= y (max2 x y))))
 (check-synth)
```

Invariant Generation as SyGuS

```
bool x, y, z
int  a, b, c

  while( Test ) {
    loop-body
    ….

}
```

- Goal: Find inductive loop invariant automatically

- Function to be synthesized

  Inv (bool x, bool z, int a, int b) : bool

- Compile loop-body into a logical predicate

  Body(x,y,z,a,b,c, x',y',z',a',b',c')

- Specification:

  Inv & Body & Test' $\Rightarrow$ Inv'

- Template for set of candidate invariants

  Term := a | b | Const | Term + Term | If-Then-Else (Cond, Term, Term)

  Cond := x | z | Cond & Cond | ~ Cond | (Cond)

Safety Verification of Transition Systems

- Symbolic Transition System S
  1. Set X of typed state variables
  2. Initial states given by formula Init(X)
  3. Transition relation given by formula Trans(X,X')

- Safety verification problem: Given a property φ(X), show that every reachable state of S satisfies φ

- Solution 1: Compute set of reachable states of S by iterated fixpoint

- Solution 2: Find inductive invariant separating initial and bad states

- Formalized as SyGuS problem: Synthesize Inv(X) s.t.
  1. Init(X) -> Inv(X)
  2. Inv(X) -> ~φ(X)
  3. Inv(X) & Trans(X,X') -> Inv(X')

Program Optimization as SyGuS

- Type matrix: 2x2 Matrix with Bit-vector[32] entries

  Theory: Bit-vectors with arithmetic

- Function to be synthesized f(matrix A, B) : matrix

- Specification: f(A,B) is matrix product

  f(A,B)[1,1] = A[1,1]*B[1,1] + A[1,2]*B[2,1]

  …

- Set of candidate implementations

  Expressions with at most 7 occurrences of *

  Unrestricted use of +

  let expressions allowed

- Benefit of saving this one multiplication: Strassen's $O(n^{2.87})$ algorithm for matrix multiplication

- Can we use only 6 multiplication operations?

Program Sketching as SyGuS

- Sketch programming system
  - C program P with ?? (holes)
  - Find expressions for holes so as to satisfy assertions
- Each hole corresponds to a separate function symbol
- Specification: P with holes filled in satisfies assertions
  - Loops/recursive calls in P need to be unrolled fixed no of times
- Set of candidate implementations for each hole:
  - All type-consistent expressions
- Not yet explored:
  - How to exploit flexibility of separation betn syntactic and semantic constraints for computational benefits?

SyGuS Benchmarks

- Over 500 benchmarks (see www.sygus.org)
- Hacker's Delight: Tricky bit-vector manipulation programs
- Invariant generation: From software verification competition
- Robotic controller: Autonomous vehicle routing
- ICFP Programming competition
- Competition of solvers (held at FLoC 2014)

Synthesis Puzzle 2: Cinderella v. stepmother

There are five buckets arranged in a circle. Each bucket can hold upto B liters of water. Initially all buckets are empty. The wicked stepmother and Cinderella take turns playing the following game:

Stepmother brings 1 liter of additional water and splits it into 5 buckets.
If any of the buckets overflows, stepmother wins the game.
If not, Cinderella gets to empty two adjacent buckets. If the game goes on forever, Cinderella wins.

Find B* such that if B < B* the stepmother has a winning strategy, and if B = B*, Cinderella has a winning strategy.
And give a proof that your strategies work!

Reference: Bodlaender et al, IFIP TCS 2012

Outline of Lectures

- ✓ Program Verification and SMT Solvers
- ✓ Motivation for Syntax-Guided Synthesis (SyGuS)
- ✓ Formalization of SyGuS
- ❖ Solving SyGuS
- ❑ TRANSIT for Protocol Specification

Solving SyGuS

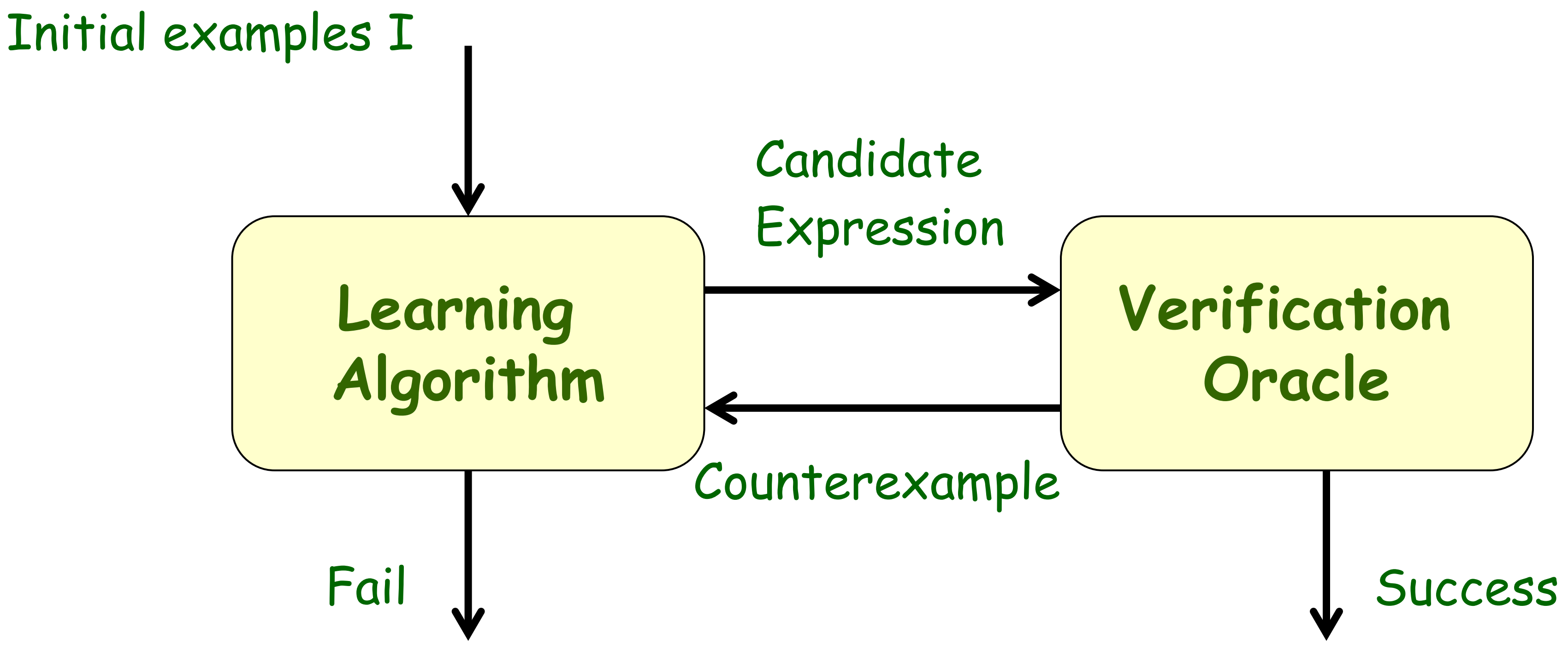
- Is SyGuS same as solving SMT formulas with quantifier alternation?

- SyGuS can sometimes be reduced to Quantified-SMT, but not always
  - Set E is all linear expressions over input vars x, y
    SyGuS reduces to Exists a,b,c. Forall X. φ [f/ ax+by+c]
  - Set E is all conditional expressions
    SyGuS cannot be reduced to deciding a formula in LIA

- Syntactic structure of the set E of candidate implementations can be used effectively by a solver

- Existing work on solving Quantified-SMT formulas suggests solution strategies for SyGuS

SyGuS as Active Learning



Concept class: Set E of expressions

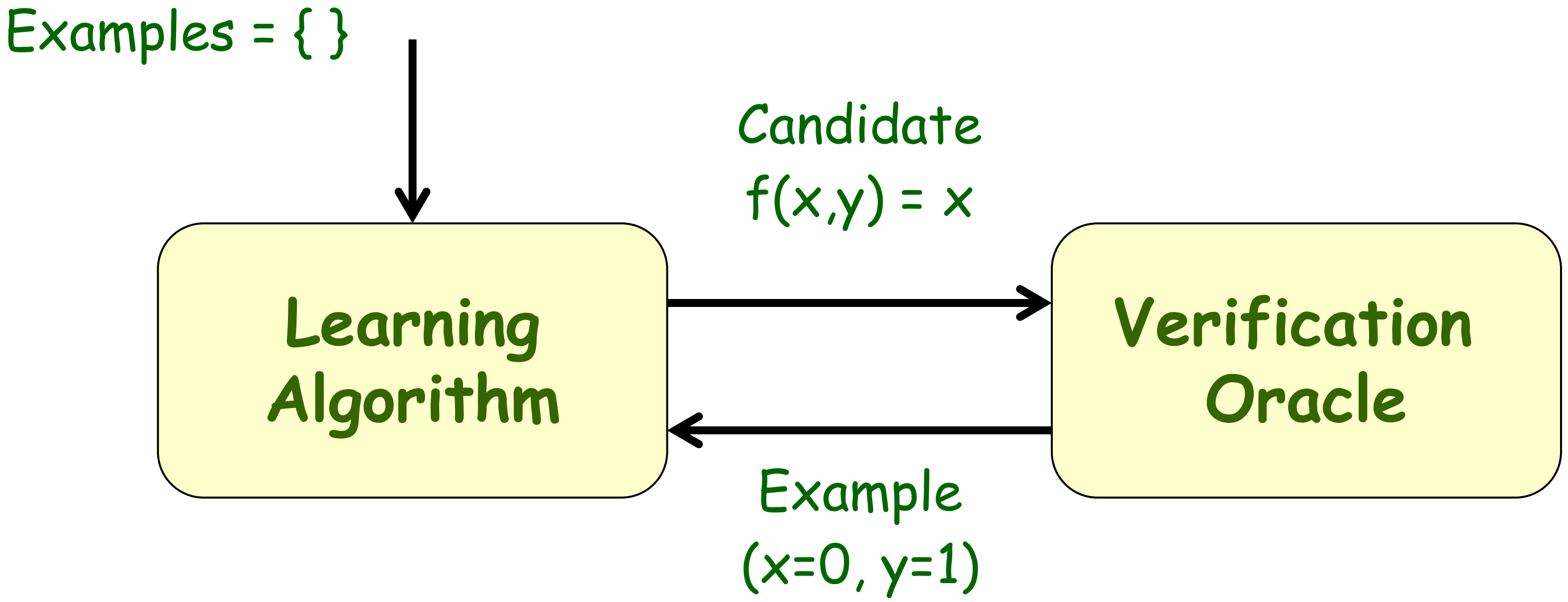
Examples: Concrete input values

Counter-Example Guided Inductive Synthesis

- Concrete inputs I for learning f(x,y) = { (x=a,y=b), (x=a',y=b'), ….}
- Learning algorithm proposes candidate expression e such that φ[f/e] holds for all values in I
- Check if φ [f/e] is valid for all values using SMT solver
- If valid, then stop and return e
- If not, let (x=α, y=β, ….) be a counter-example (satisfies ~ φ[f/e])
- Add (x=α, y=β) to tests I for next iteration

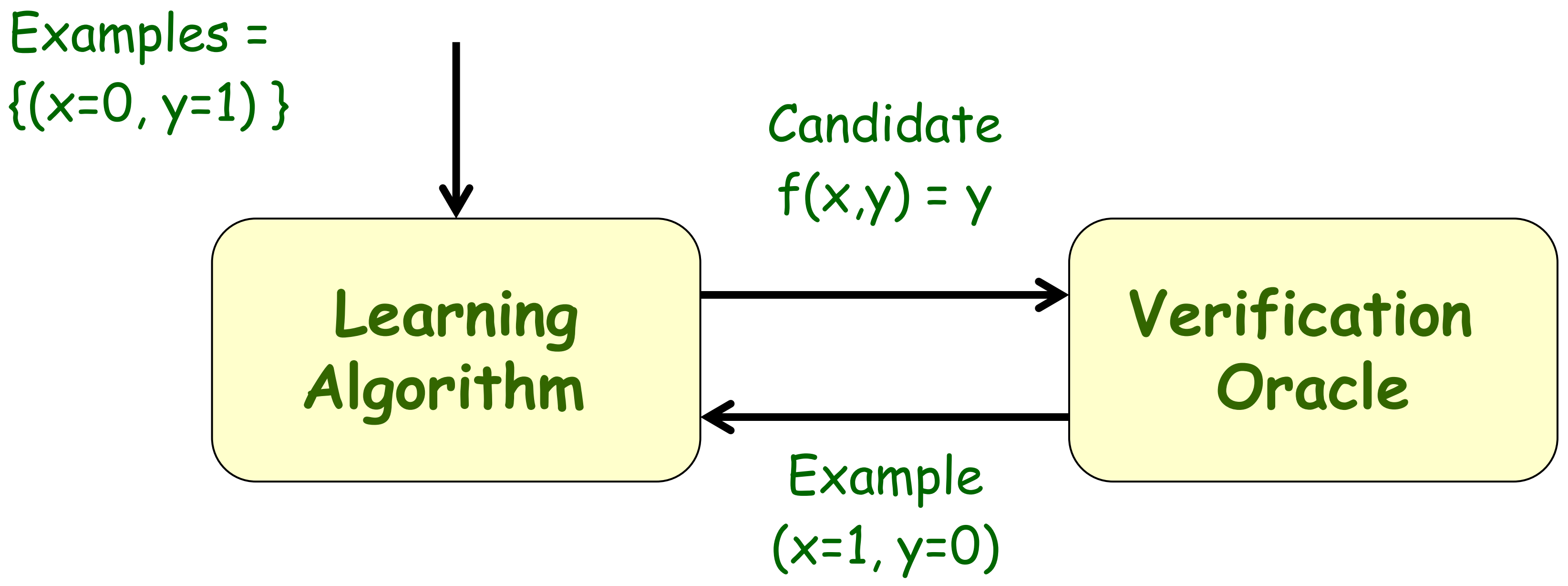
CEGIS Example

- Specification: $(x \leq f(x,y))$ & $(y \leq f(x,y))$ & $(f(x,y) = x \mid f(x,y) = y)$
- Set E: All expressions built from x,y,0,1, Comparison, +, If-Then-Else



CEGIS Example

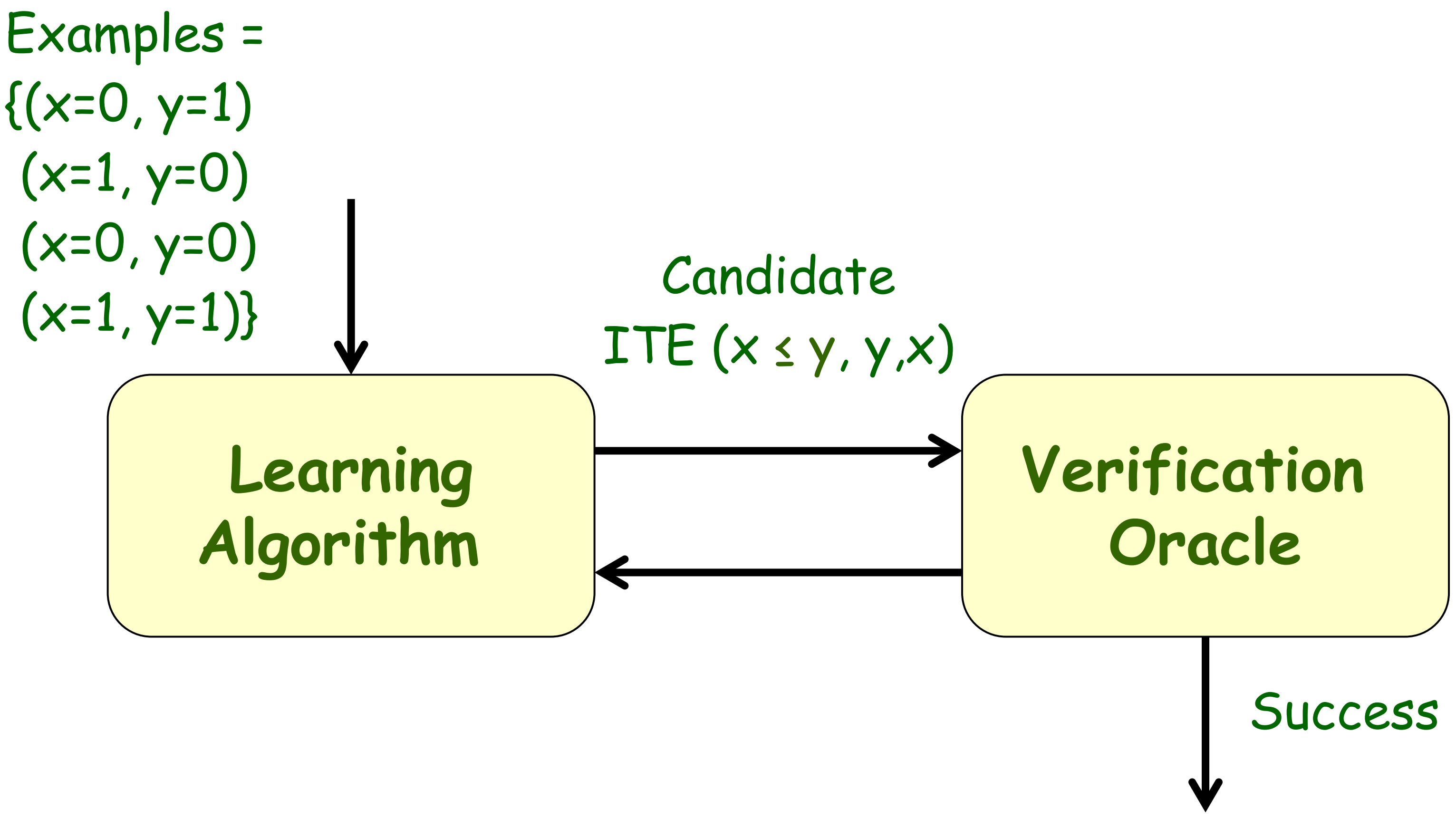
- Specification: $(x \le f(x,y))$ & $(y \le f(x,y))$ & $(f(x,y) = x \mid f(x,y) = y)$
- Set E: All expressions built from x,y,0,1, Comparison, +, If-Then-Else



CEGIS Example

- Specification: (x ≤ f(x,y)) & (y ≤ f(x,y)) & (f(x,y) =x | f(x,y)=y)

- Set E: All expressions built from x,y,0,1, Comparison, +, If-Then-Else



SyGuS Solutions

- CEGIS approach (Solar-Lezama et al, ASPLOS'08)

- Similar strategies for solving quantified formulas and invariant generation

- Learning strategies based on:
  - Enumerative (search with pruning): Udupa et al (PLDI'13)
  - Symbolic (solving constraints): Gulwani et al (PLDI'11)
  - Stochastic (probabilistic walk): Schkufza et al (ASPLOS'13)

Enumerative Learning

- Find an expression consistent with a given set of concrete examples

- Enumerate expressions in increasing size, and evaluate each expression on all concrete inputs to check consistency

- Key optimization for efficient pruning of search space:

  Expressions e_1 and e_2 are equivalent

  if $e_1(a,b)=e_2(a,b)$ on all concrete values $(x=a,y=b)$ in Examples

  Only one representative among equivalent subexpressions needs to be considered for building larger expressions

- Fast and robust for learning expressions with ~ 15 nodes

Enumerative CEGIS

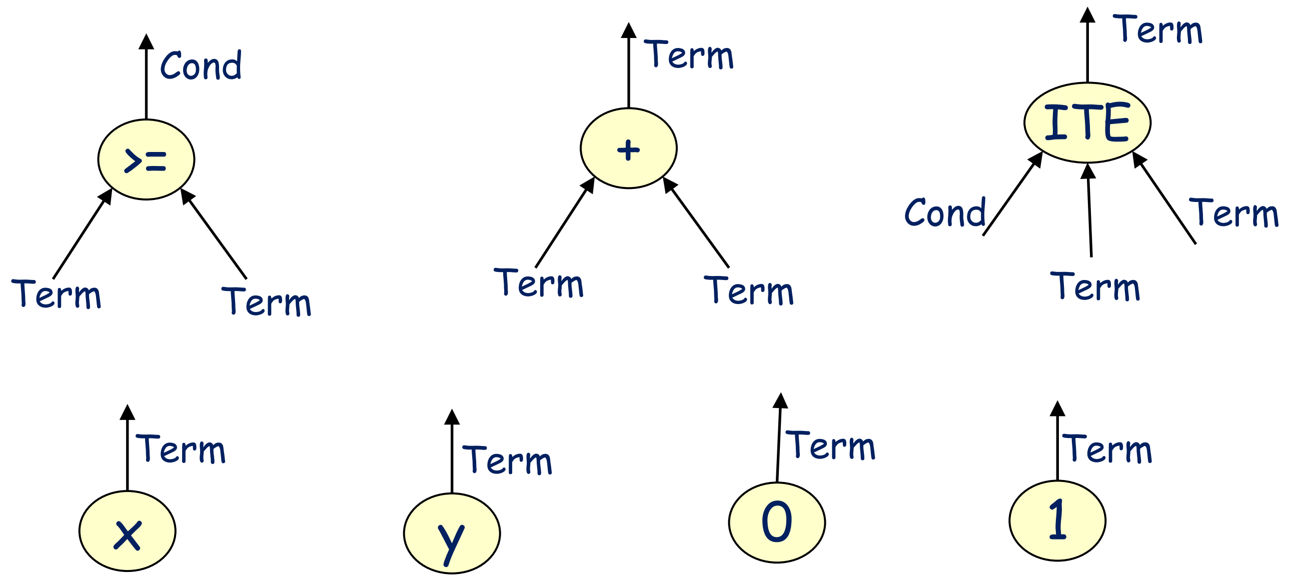
Synthesized Expr.	Counter-Example
x	(x=0, y =1)
y	(x=1, y =0)
1	(x=0, y =0)
x + y	(x=1, y =1)
ITE(x<y,y,x)	Verified

200 expressions searched
4 verifier calls, 80 stored expressions

Symbolic Learning

- Use a constraint solver for both the synthesis and verification step.

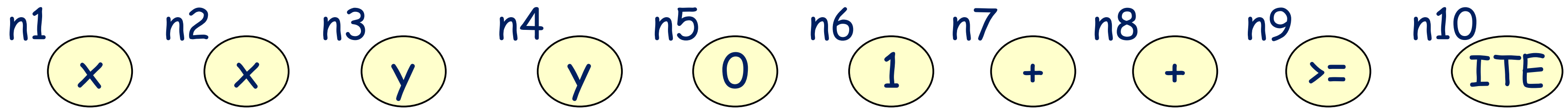
- Each production in the grammar is thought of as a component.
  Input and Output ports of every component are typed.



- A well-typed loop-free program comprising these component corresponds to an expression DAG from the grammar.

Symbolic Learning

- Start with a library consisting of some number of occurrences of each component.



- Synthesis Constraints:

  Shape is a DAG, Types are consistent

  Spec φ[f/e] is satisfied on every concrete input values in I

- Use an SMT solver (Z3) to find a satisfying solution.

- If synthesis fails, try increasing the number of occurrences of components in the library in an outer loop

Symbolic CEGIS

Synthesized Expr.	Counter-Example
$o_1 = x$	$(x=-1, y=0)$
$o_1 = x < x$ $o_2 = ITE(o_1, y, x)$	$(x=0, y=-1)$
$o_1 = y > x$ $o_2 = ITE(o_1, y, x)$	Verified

1 instance of each library operator

Stochastic Learning

- Idea: Find desired expression e by probabilistic walk on graph where nodes are expressions and edges capture single-edits

- Metropolis-Hastings Algorithm: Given a probability distribution P over domain X, and an ergodic Markov chain over X, samples from X

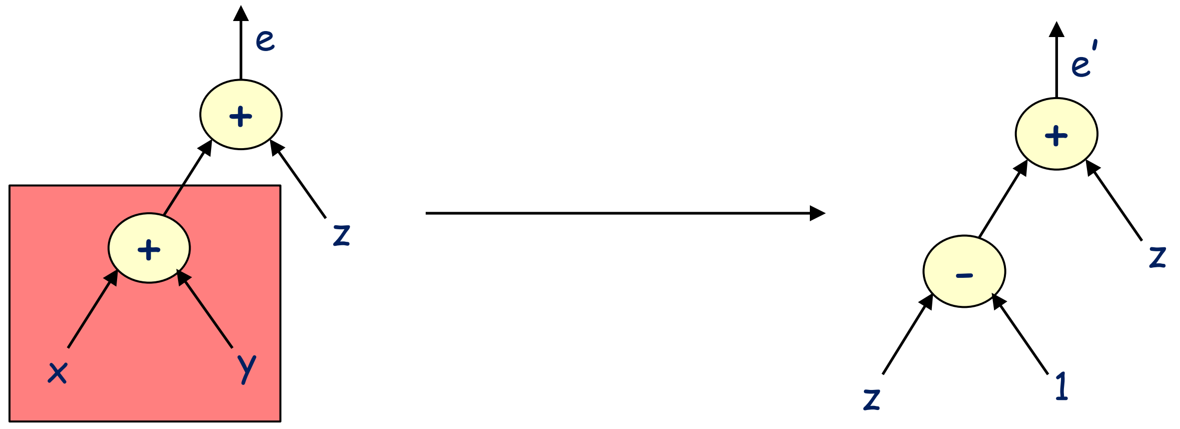
- Fix expression size n. X is the set of expressions E_n of size n. P(e) $\propto$Score(e) ("Extent to which e meets the spec φ")

- For a given set I of concrete inputs, Score(e) = exp(- 0.5 Wrong(e)), where Wrong(e) = No of examples in I for which ~ φ [f/e]

- Score(e) is large when Wrong(e) is small. Expressions e with Wrong(e) = 0 more likely to be chosen in the limit than any other expression

Stochastic Learning

- Initial candidate expression e sampled uniformly from E_n
- When Score(e) = 1, return e
- Pick node v in parse tree of e uniformly at random. Replace subtree rooted at e with subtree of same size, sampled uniformly



- With probability min{ 1, Score(e')/Score(e) }, replace e with e'
- Outer loop responsible for updating expression size n

Stochastic CEGIS

Let n = 6 (786 possible expressions of size 6)

CEXs = {(-1,-4), (-1,-3), (-1,-2), (1,1), (1,2)}

$e = \mathrm{ITE}(x<0, y, x)$ $\qquad p_I(e) = 1/768$

$e' = \mathrm{ITE}(y<0, y, x)$ $\qquad p_C(e') = 1/6 * 1/48$

$$p_M(e \rightarrow e') = \min(1, \mathrm{Score}(e')/\mathrm{Score}(e)) = \exp(-0.5)$$

$e'' = \mathrm{ITE}(x<y, y, x)$ $\qquad p_C(e'') = 1/6 * 1/48$

$$p_M(e \rightarrow e'') = \min(1, \mathrm{Score}(e'')/\mathrm{Score}(e)) = 1$$

Benchmarks and Implementation

- Prototype implementation of Enumerative/Symbolic/Stochastic CEGIS
- Benchmarks:
  - Bit-manipulation programs from Hacker's delight
  - Integer arithmetic: Find max, search in sorted array
  - Challenge problems such as computing Morton's number
- Multiple variants of each benchmark by varying grammar
- Results are not conclusive as implementations are unoptimized, but offers first opportunity to compare solution strategies

Evaluation

- Enumerative CEGIS has best performance, and solves many benchmarks within seconds

  Potential problem: Synthesis of complex constants

- Symbolic CEGIS is unable to find answers on most benchmarks

  Caveat: Sketch succeeds on many of these

- Choice of grammar has impact on synthesis time

  When E is set of all possible expressions, solvers struggle

- None of the solvers succeed on some benchmarks

- Bottomline: Improving solvers is a great opportunity for research !

Stepmother wins if B<2

Round 1:

Stepmother: Add 0.5 lit to buckets 1 and 3

Cinderella: Empty one of the buckets, say third

Round 2:

Stepmother: Add 0.25 lit to bucket 1 and 0.75 lit to bucket 3

Cinderella: Empty bucket 3

…

After n rounds, bucket 1 contains $1 - 1/2^n$ lit of water

If $B < 2$, then after some N rounds bucket 1 contains more than B-1 lit of water, stepmother can win in (N+1)th round by adding 1 lit to it

Cinderella wins if B=2

Cinderella maintains the following invariant:

$$(a1 + a3 < 1) \;\&\; (a2 <= 1) \;\&\; (a4 = 0) \;\&\; (a5 = 0)$$

a1, a2, a3, a4, a5: water quantities starting at some bucket

If this condition holds after n rounds, stepmother cannot win in the next round. Thus, if this is an invariant, then Cinderella wins.

Invariant holds initially.

Assume the invariant holds at the beginning of a round.

Goal: Cinderella can enforce the invariant, no matter what the stepmother does, after her own turn.

Cinderella wins if B=2

At the beginning of the round, we have:

$(a1 + a3 < 1)$ & $(a2 <= 1)$ & $(a4 = 0)$ & $(a5 = 0)$

b1, b2, b3, b4, b5: water quantities after stepmother's turn

Claim: $b1 + b3 + b4 + b5 < 2$

Either $(b1 + b4 < 1)$ or $(b3 + b5 < 1)$

Suppose $(b1 + b4 < 1)$. Other case similar.

Cinderella strategy: empty buckets 2 and 3.
We have: $(b4 + b2 < 1)$ & $(b5 <= 1)$ & $(b2 = 0)$ & $(b3 = 0)$

SyGuS Recap

www.sygus.org

- Contribution: Formalization of syntax-guided synthesis problem
  - Not language specific such as Sketch, Scala^Z3,…
  - Not as low-level as (quantified) SMT

- Advantages compared to classical synthesis
  1. Set E can be used to restrict search (computational benefits)
  2. Programmer flexibility: Mix of specification styles
  3. Set E can restrict implementation for resource optimization
  4. Beyond deductive solution strategies: Search, inductive inference

- Prototype implementation of 3 solution strategies

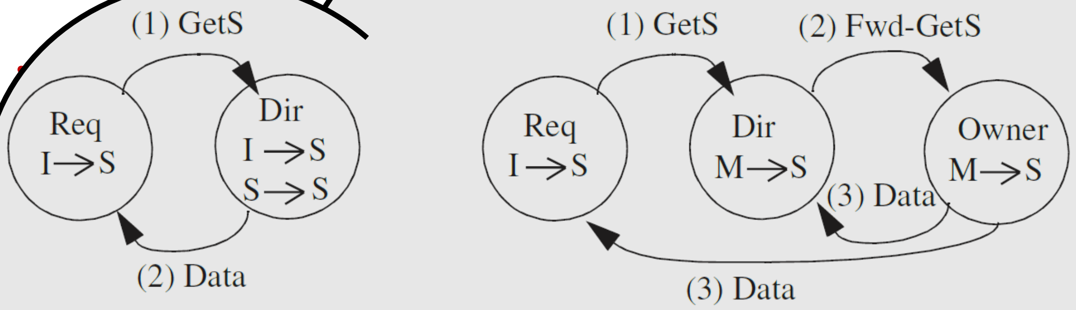
- Initial set of benchmarks, competition (held in FLoC 2014), and evaluation (Winner: Enumerative CEGIS)

Outline of Lectures

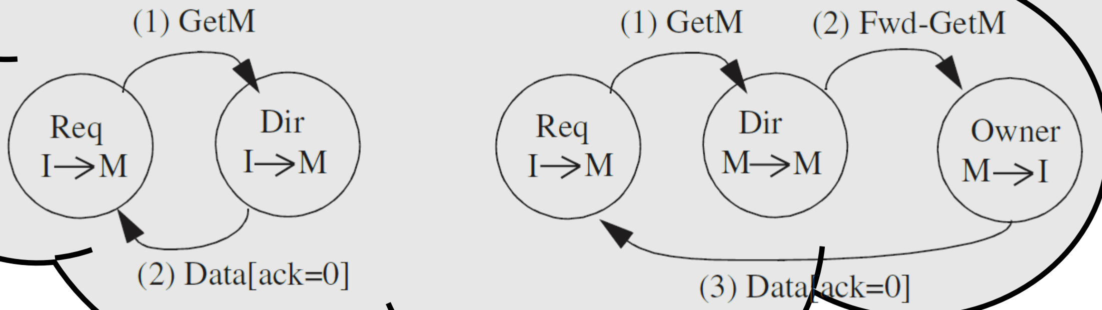
- ✓ Program Verification and SMT Solvers
- ✓ Motivation for Syntax-Guided Synthesis (SyGuS)
- ✓ Formalization of SyGuS
- ✓ Solving SyGuS
- ❖ TRANSIT for Protocol Specification

Protocols

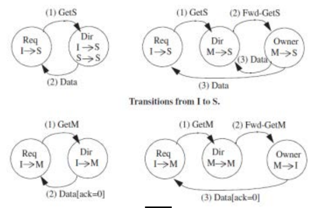
- Design challenging due to asynchronous model of communication
- Examples: Cache coherence protocols, Distributed coordination algorithms
- Successful application domain for model checking
  - SPIN: Distributed algorithms
  - Murphi, SMV: Hardware protocols
  - Industrial adoption (Intel, IBM, …)
- Correctness involves both safety and liveness properties



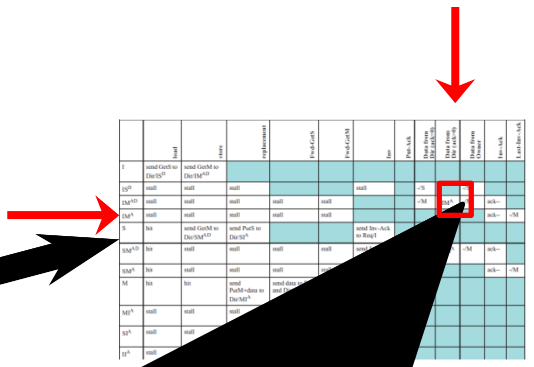
Transitions from I to S.



Traditional Specifications

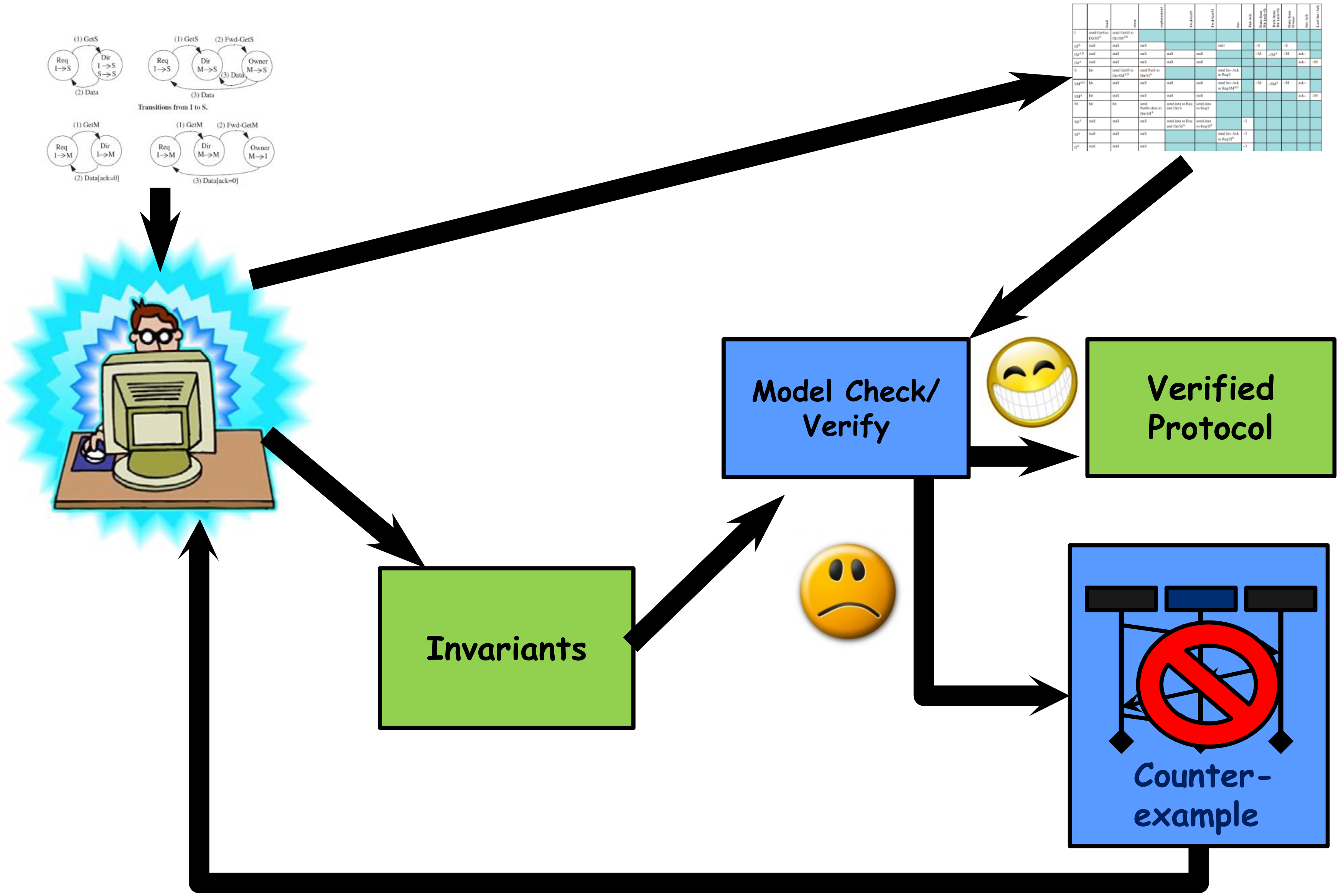


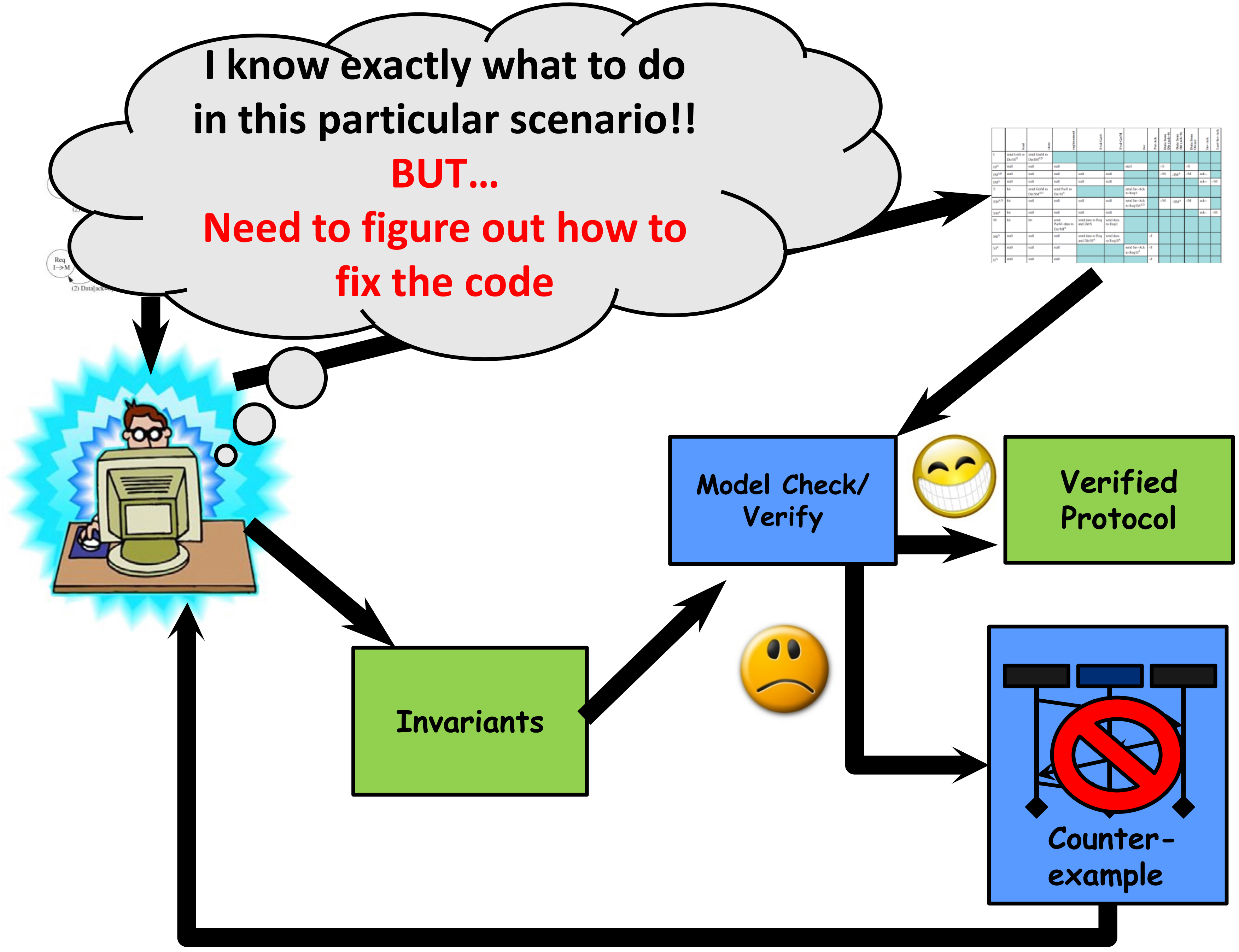
Transitions from I to S.

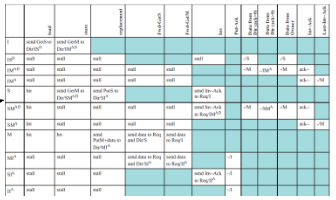


```
State = D_BUSY & InMsg.MType = UNBLOCK_S ==>
Begin
    State = D_M;
    sharers = SetUnion(Sharers, SetOf(InMsg.Sender));
    SendMsg({Type = ACK,
             Acks = 1,
             InMsg.Sender});
EndRule;
```

Traditional Specifications



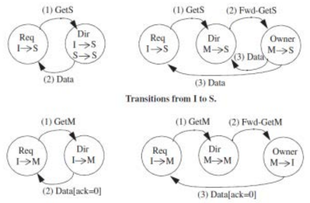

I know exactly what to do
in this particular scenario!!
BUT…
Need to figure out how to
fix the code
Model Check/
Verify
Verified
Protocol
Invariants
Counter-
example

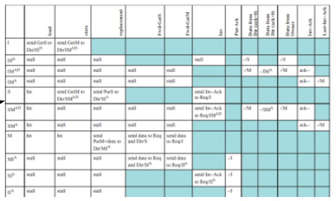


	load	store	replacement	Fwd-GetS	Fwd-GetM	Inv	Put-Ack	Data from Dir (ack=0)	Data from Dir (ack>0)	Data from Owner	Inv-Ack	Last-Inv-Ack
I	send GetS to Dir/IS^D	send GetM to Dir/IM^{AD}										
IS^D	stall	stall	stall			stall		-/S		-/S		
IM^{AD}	stall	stall	stall	stall	stall			-/M	-/IM^A	-/M	ack--	
IM^A	stall	stall	stall	stall	stall						ack--	-/M
S	hit	send GetM to Dir/SM^{AD}	send PutS to Dir/SI^A			send Inv-Ack to Req/I						
SM^{AD}	hit	stall	stall	stall	stall	send Inv-Ack to Req/IM^{AD}		-/M	-/SM^A	-/M	ack--	
SM^A	hit	stall	stall	stall	stall						ack--	-/M
M	hit	hit	send PutM+data to Dir/MI^A	send data to Req and Dir/S	send data to Req/I							
MI^A	stall	stall	stall	send data to Req and Dir/SI^A	send data to Req/II^A		-/I					
SI^A	stall	stall	stall			send Inv-Ack to Req/II^A	-/I					
II^A	stall	stall	stall				-/I					

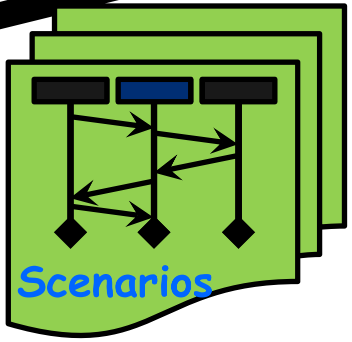
Can we make the process of specifying distributed protocols easier?

TRANSIT Specification





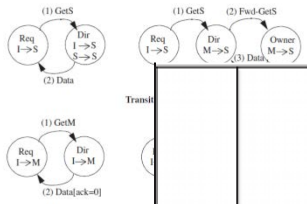
	load	store	replacement	Fwd-GetS	Fwd-GetM	Inv	Put-Ack	Data from Dir (ack=0)	Data from Dir (ack>0)	Data from Owner	Inv-Ack	Last-Inv-Ack
I	send GetS to Dir/IS^D	send GetM to Dir/IM^{AD}										
IS^D	stall	stall	stall			stall		-/S		-/S		
IM^{AD}	stall	stall	stall	stall	stall			-/M	-/IM^A	-/M	ack--	
IM^A	stall	stall	stall	stall	stall						ack--	-/M
S	hit	send GetM to Dir/SM^{AD}	send PutS to Dir/SI^A			send Inv-Ack to Req/I						
SM^{AD}	hit	stall	stall	stall	stall	send Inv-Ack to Req/IM^{AD}		-/M	-/SM^A	-/M	ack--	
SM^A	hit	stall	stall	stall	stall						ack--	-/M
M	hit	hit	send PutM+data to Dir/MI^A	send data to Req and Dir/S	send data to Req/I							
MI^A	stall	stall	stall	send data to Req and Dir/SI^A	send data to Req/II^A		-/I					
SI^A	stall	stall	stall			send Inv-Ack to Req/II^A	-/I					
II^A	stall	stall	stall				-/I					



Scenarios:

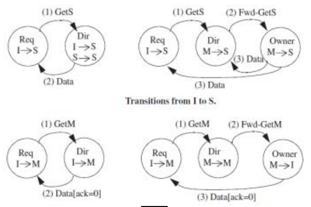
- **Describe execution traces**
- **Translated from informal specs**
- **Can be symbolic**
- **Can be concrete**

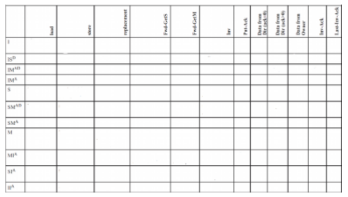
TRANSIT Specification

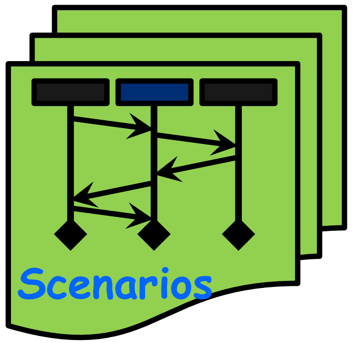


	load	store	replacement	Fwd-GetS	Fwd-GetM	Inv	Put-Ack	Data from Dir (ack=0)	Data from Dir (ack>0)	Data from Owner	Inv-Ack	Last-Inv-Ack
I												
IS^D												
IM^{AD}												
IM^A												
S												
SM^{AD}												
SM^A												
M												
MI^A												
SI^A												
II^A												

TRANSIT Specification

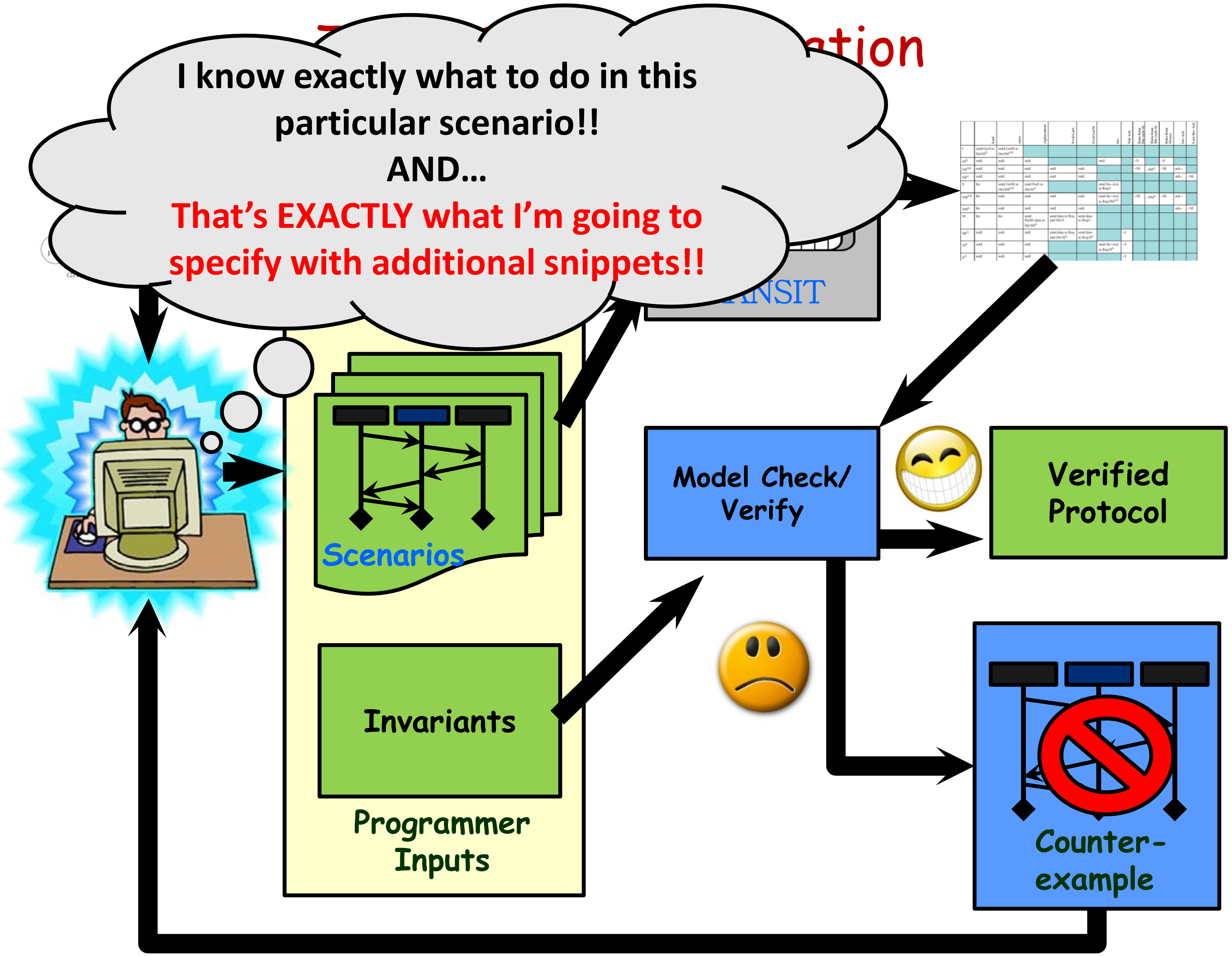






Protocol Skeleton:

- Communication architecture
- Message types
- Set of processes
- State variables for processes



I know exactly what to do in this particular scenario!!

AND…

That's EXACTLY what I'm going to specify with additional snippets!!

Goal

Protocol Skeleton

+

Scenarios

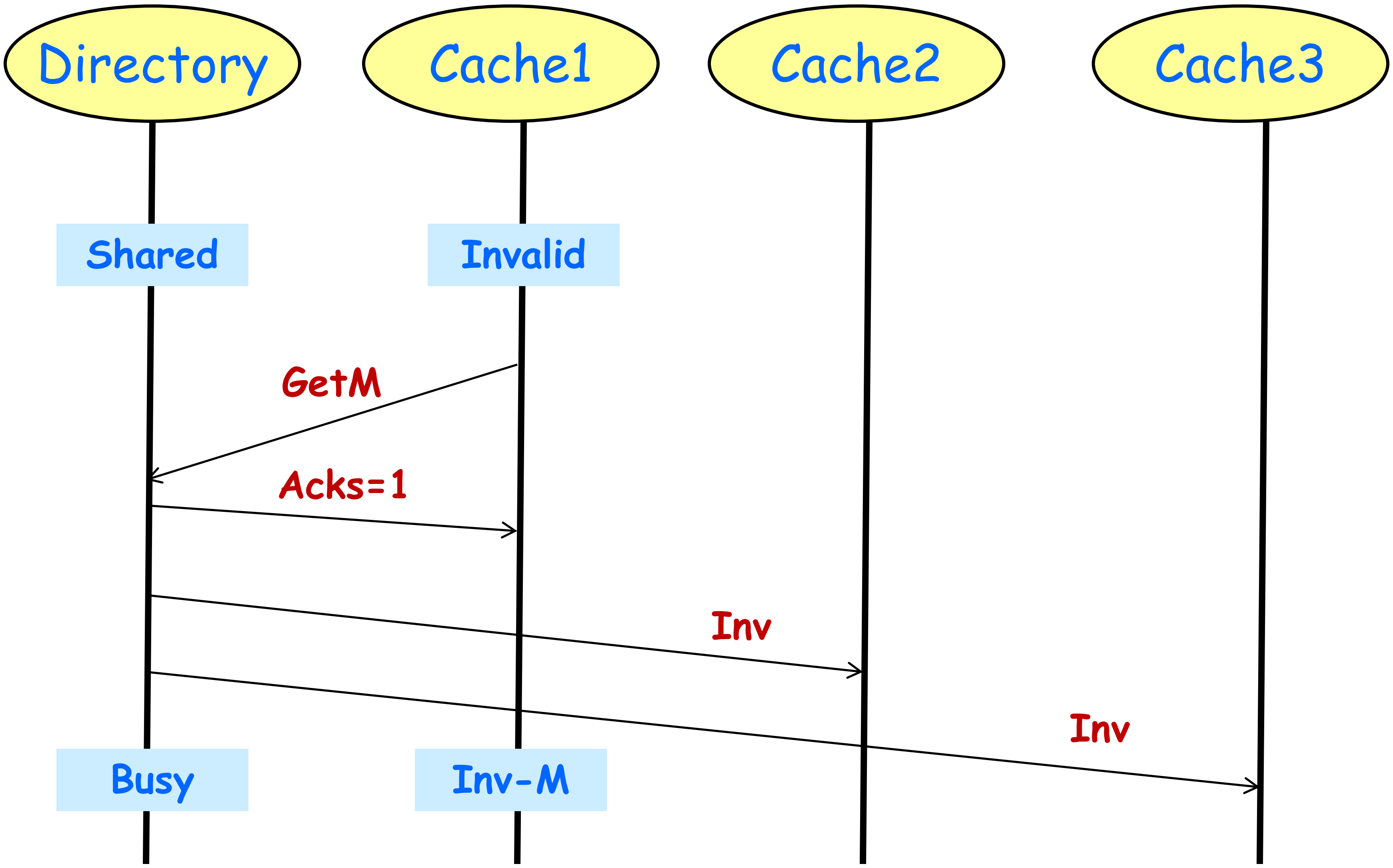
+

Invariants

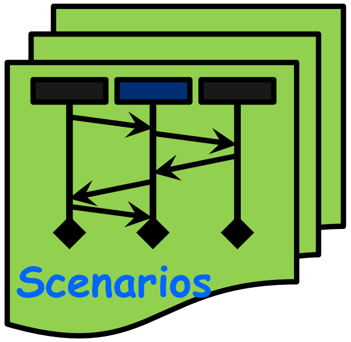
⬇

Completed Protocol Specification

Example Scenario



Snippets and Scenarios



- Scenarios:
  - Sequence of message exchanges/transitions
  - Transcribed from informal specs
  - A collection of *snippets*

- Snippets:
  - Describe actions on a *single* transition
  - Relate current values of variables to updated values
  - Can be concrete or symbolic (Concolic)

Concolic Snippet Example 1

```
Process Directory
Transition
        From Shared
        To Busy
        Input channel:  ReqMsg
        Output channels: RespMsg, InvMsg

        Guard:
          ReqMsg.Sender = 1 & ReqMsg.Type = GetM & Sharers ={2,3}

        Update:
           RespMsg.Acks = 2;
           RespMsg.Destination = 1;
           InvMsg.Destination = {2,3}
```

Values of all variables can be concrete

Concolic Snippet Example 2

```
Process Directory
Transition
        From Shared
        To Busy
        Input channel:  ReqMsg
        Output channels: RespMsg, InvMsg

        Guard:
          ReqMsg.Type = GetM & ReqMsg.Sender = 1

        Update:
           RespMsg.Acks =  Size ( Sharers );
           RespMsg.Destination = 1;
           InvMsg.Destination =  Sharers
```

Same snippet can mix concrete and symbolic values

Concolic Snippet Example 3

```
Process Directory
Transition
          From Shared
          To Busy
          Input channel:  ReqMsg
          Output channels: RespMsg, InvMsg

          Guard:
             ReqMsg.Type = GetM

          Update:
             RespMsg.Acks =  Size ( Sharers );
             RespMsg.Destination = ReqMsg.Sender;
             InvMsg.Destination =  Sharers – ReqMsg.Sender
```

Values of all variables can be symbolic:
Classical EFSM description maps directly to such snippets

From Snippets to Transition Code

- To generate a completed protocol TRANSIT needs to:
- *Find guards and updates consistent with given snippets*

```
Rule

==>
Begin
    State =
    sharers =
    SendAck({ });
EndRule;
```

- Expression grammar: Int, Bool, BitVector types
- Arithmetic, bit-vector and conditional operations
- Finding desired expressions exactly the SyGuS problem
- Enumerative CEGIS solver

Iterative Design

```
From Shared  To Busy
 Input channel:  ReqMsg
Output channels: RespMsg, InvMsg
    Guard:
        ReqMsg.Sender = 1 & ReqMsg.Type = GetM & Sharers ={2,3}
    Update:
         RespMsg.Acks = 2;
         InvMsg.Destination = {2,3}
```

Based on this single example, synthesis tool computes update:

```
RespMsg.Acks = Size(Sharers);
InvMsg.Destination = Sharers
```

But this is incorrect, and protocol deadlocks

Iterative Design Continued

Designer adds another concrete example
(corresponds to case when Sender is a Sharer)

```
From Shared  To Busy
 Input channel:  ReqMsg
Output channels: RespMsg, InvMsg
   Guard:
      ReqMsg.Sender = 1 & ReqMsg.Type = GetM & Sharers ={1,2}
   Update:
       RespMsg.Acks = 1;
       InvMsg.Destination = {2}
```

Based on two snippets, synthesis tool computes update:

```
RespMsg.Acks = Size(Sharers + ReqMsg.Sender) - 1;
InvMsg.Destination = Sharers – ReqMsg.Sender
```

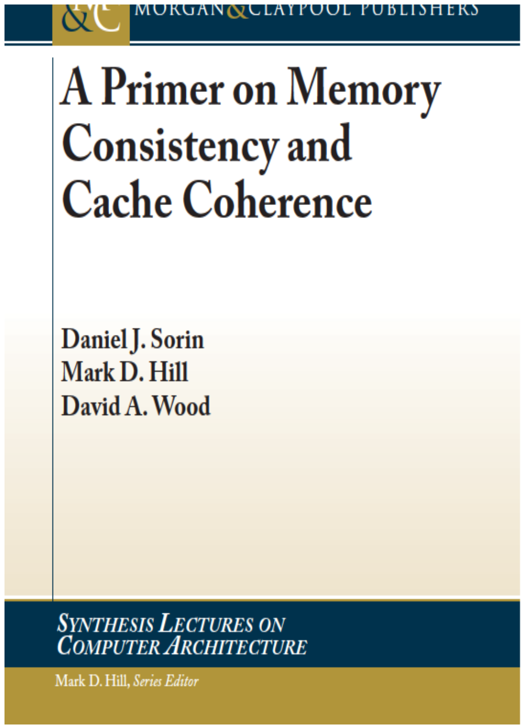
Implementation Evaluation

- Starting point: cache coherence protocols described in
  A Primer on memory Consistency and Cache Coherence
  Sorin, Hill, Wood, 2011
- Translated EFSMs for 2 protocols into (mostly concrete) snippets

	VI Protocol	MSI Protocol
Snippets used	19	77
Update expressions synthesized	49	157
Guard expressions synthesized	17	45
Expressions explored	3.1K	44.5K
Synthesis time	5 sec	134 sec
States explored by Murphi	140K	154K

Methodology Evaluation

- Case study: Can a user with no prior experience in designing cache coherence protocol build a correct protocol from textbook description?



MORGAN&CLAYPOOL PUBLISHERS

A Primer on Memory Consistency and Cache Coherence

Daniel J. Sorin
Mark D. Hill
David A. Wood

SYNTHESIS LECTURES ON COMPUTER ARCHITECTURE

Mark D. Hill, *Series Editor*

- Case study A: Version of MSI protocol with non-blocking directory progress

- Case study B: Augmenting MSI protocol with E state to obtain MESI

Sample Transitions

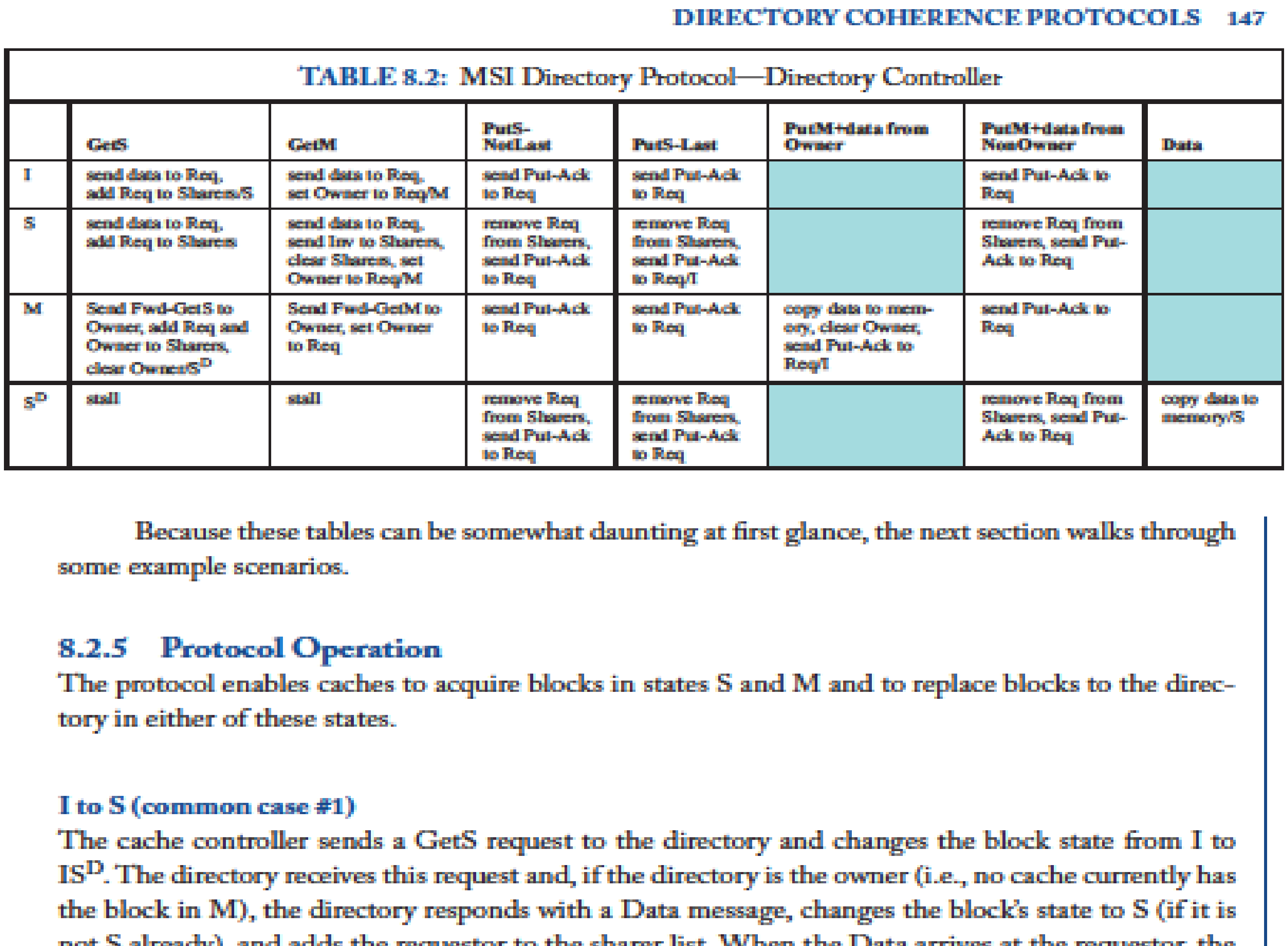


TABLE 8.2: MSI Directory Protocol—Directory Controller

	GetS	GetM	PutS-NotLast	PutS-Last	PutM+data from Owner	PutM+data from NonOwner	Data
I	send data to Req, add Req to Sharers/S	send data to Req, set Owner to Req/M	send Put-Ack to Req	send Put-Ack to Req		send Put-Ack to Req	
S	send data to Req, add Req to Sharers	send data to Req, send Inv to Sharers, clear Sharers, set Owner to Req/M	remove Req from Sharers, send Put-Ack to Req	remove Req from Sharers, send Put-Ack to Req/I		remove Req from Sharers, send Put-Ack to Req	
M	Send Fwd-GetS to Owner, add Req and Owner to Sharers, clear Owner/S^D	Send Fwd-GetM to Owner, set Owner to Req	send Put-Ack to Req	send Put-Ack to Req	copy data to memory, clear Owner, send Put-Ack to Req/I	send Put-Ack to Req	
S^D	stall	stall	remove Req from Sharers, send Put-Ack to Req	remove Req from Sharers, send Put-Ack to Req		remove Req from Sharers, send Put-Ack to Req	copy data to memory/S

Because these tables can be somewhat daunting at first glance, the next section walks through some example scenarios.

8.2.5 Protocol Operation

The protocol enables caches to acquire blocks in states S and M and to replace blocks to the directory in either of these states.

I to S (common case #1)

The cache controller sends a GetS request to the directory and changes the block state from I to IS^D. The directory receives this request and, if the directory is the owner (i.e., no cache currently has the block in M), the directory responds with a Data message, changes the block's state to S (if it is

Sample Scenarios

Transition from I to S

DIRECTORY COHERENCE PROTOCOLS 143

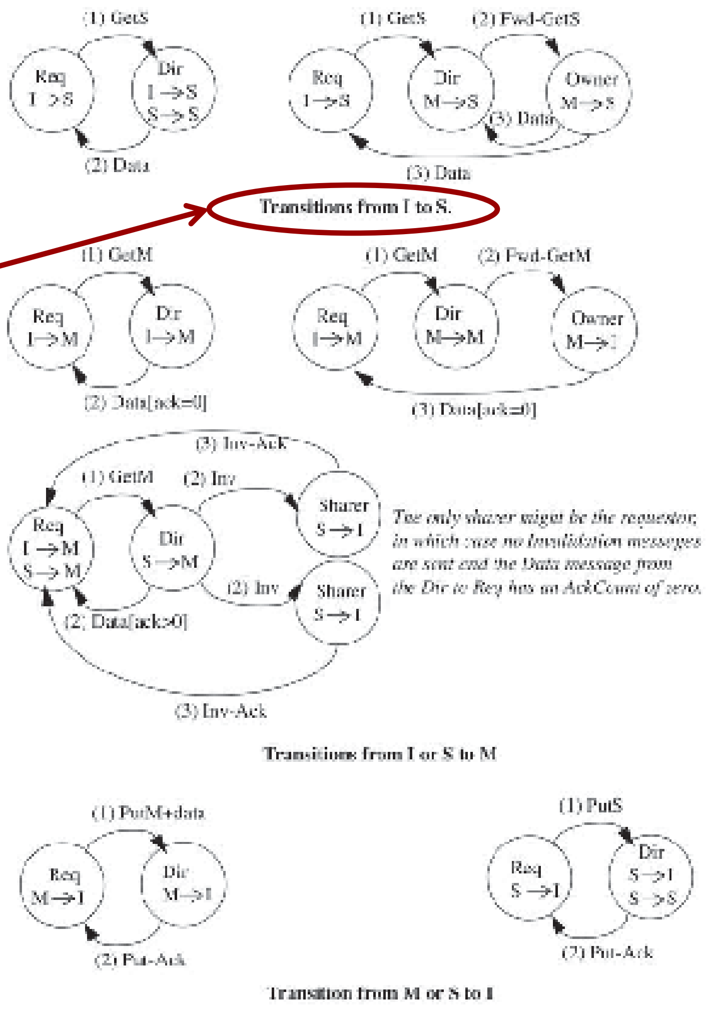


FIGURE 8.3: High-Level Description of MSI Directory Protocol. In each transition, the cache controller that requests the transaction is denoted "Req".

sharers encoded as a one-hot bit vector (if the block is in state S). We illustrate a directory entry in Figure 8.2. In Section 8.5, we will discuss other encodings of directory entries.

Before presenting the detailed specification, we first illustrate a higher level abstraction of the protocol in order to understand its fundamental behaviors. In Figure 8.3, we show the transactions in which a cache controller issues coherence requests to change permissions from I to S, I or S to

Experience Report

	Case study A	Case study B
Snippets used in first version	19	96
Time to implement first version	2 hrs	6 hrs
Snippets used in final version	86	108
Number of iterations	13	8
Total manual effort	6 hrs	13 hrs
Number of counterexamples examined	5	6
Synthesis time in last iteration	52 min	15 min
Number of update expressions synthesized	175	260
Number of guard expressions synthesized	80	74
States explored in final protocol	7.7 million	1.5 million

Usability: SGI Origin

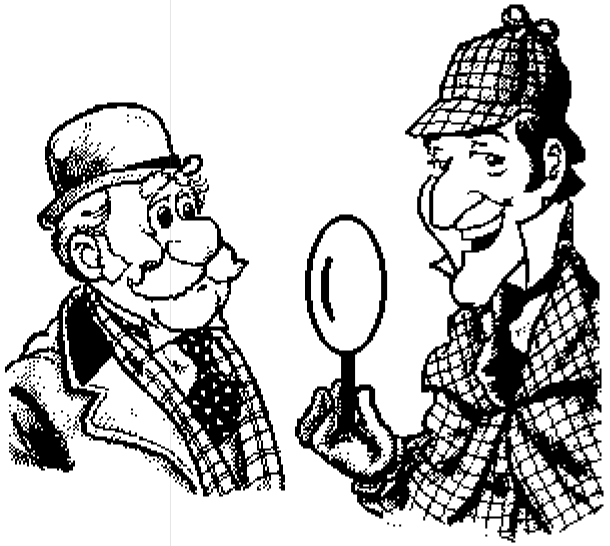
- Specified the protocol used in SGI Origin (complex, industrial protocol)
- Intermixed concrete and symbolic snippets
- Successfully converged to a correct protocol
- Computational effort: final synthesis took 30 minutes of CPU time

Recap: TRANSIT for Protocol Design

- Specification: Protocol skeleton + Scenarios (with concolic snippets) + Invariants
- Computational effort needed to “complete” the protocol is not high
- Case studies suggest that, at least to translate text-book descriptions to working implementations, the proposed methodology helps designer
- “Model checker and SyGuS solver in the loop” is plausible for design environments
- Work in Progress:
  - Incorporate liveness properties
  - Improved solvers for completion
  - Infer auxiliary states
  - Better feedback to designers

Synthesis 2.0

Expeditions in Computer Augmented Program Engineering



- Paradigm shift in synthesis:
  - Old: Allow more concise, high-level description
  - New: Designer uses multiple, natural formats,
    - Synthesis tool assists in discovering tricky logic
- Paradigm shift in design tools:
  - Old : Any compiler transformation must be polynomial-time
  - New: Computational intractability not a show-stopper
- Common theme: Guided search in a space of programs to find one that meets multiple design goals
  - A bit like model checking, but can be interactive!