Syntax-Guided Synthesis

Rajeev Alur

Marktoberdorf Summer School 2014
Program Verification

Specification S

Program P

Verifier

Proof of correctness or Witness of a bug
Classical Program Synthesis

Specification S
High Level
"WHAT"

Synthesizer

Program P
Low Level
"HOW"
Syntax-Guided Synthesis

Specification $S$ given by logical constraints

Syntactic restrictions $R$ on the space of programs

Synthesizer

Program $P$

www.sygus.org
References

- **Syntax Guided Synthesis**
  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
Programming Technology

Libraries → Program → Compiler → Executable

- High-level programming abstractions
  (object-oriented, declarative, domain-specific..)

- Semantics-preserving transformations
  (low-level optimizations, type inference ..)

Platform
Verification Technology

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

Executable
Platform

Program
Specifications
Tests

Analysis Tool
Program Verification

- Does a program P meet its specification $\varphi$?

- 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;
}

post: ∀k : 0 ≤ k < n ⇒ A[k] ≤ 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) ⇒ (0 ≤ i < n)

And validity of formula

(0 ≤ i < n) & (i' = i+1) & (i' < n-1) ⇒ (0 ≤ i' < n)
Discharging Verification Conditions

- Check validity of
  \[(i = 0) \& (i < n-1) \Rightarrow (0 \leq i < n)\]

- Reduces to checking satisfiability of
  \[(i = 0) \& (i < n-1) \& \neg(0 \leq i < n)\]

- 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!

1960 DP ≈ 10 var
1962 DLL ≈ 10 var
1986 BDDs ≈ 100 var
1992 GSAT ≈ 300 var
1994 Hannibal ≈ 3k var
1996 GRASP ≈ 1k var
1996 Stålmarck ≈ 1000 var
1996 SATO ≈ 1k var
1996 Berkmin ≈ 10k var
1998 MiniSAT ≈ 20k var
1998 SOCRATES ≈ 300 var
2002 Berkmin ≈ 10k var
2005 MiniSAT ≈ 20k var
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)

CBMC  SAGE  VCC  Spec#

Z3  Yices  CVC4  MathSAT5
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

Program

Specifications

Tests

Analysis Tool

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

Executable

Platform

21
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

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);

When to start turning?
Backup straight
How much to turn?
Straighten

Ref: Chaudhuri, Solar-Lezama (PLDI 2010)
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

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.`
Paraglide: From Sequential to Parallel Code

Ref: Vechev et al (POPL 2010)

**Sequential Program**

```cpp
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
}
```

**Architectural Description**

- 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)

<table>
<thead>
<tr>
<th>Input</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>(425)-706-7709</td>
<td>425-706-7709</td>
</tr>
<tr>
<td>510.220.5586</td>
<td>510-220-5586</td>
</tr>
<tr>
<td>1 425 235 7654</td>
<td>425-235-7654</td>
</tr>
<tr>
<td>425 745-8139</td>
<td>425-745-8139</td>
</tr>
</tbody>
</table>

- Infers desired Excel macro program
- Iterative: user gives examples and corrections
- Incorporated in commercial version of Microsoft Excel
Given a program $P$, find a “better” equivalent program $P'$

```c
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
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: \( \forall k : 0 \leq k < n \Rightarrow A[k] \leq A[k + 1] \)
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: ∀k : 0 ≤ k < n ⇒ A[k] ≤ 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: ∀k: 0 ≤ k < n ⇒ A[k] ≤ A[k + 1]

Invariant:
∀k1, k2. 0 ≤ k1 < k2 < n ∧
        k1 < i ⇒ A[k1] ≤ A[k2]

Invariant:
    i < j ∧
i ≤ v < n ∧
(∀k1, k2. 0 ≤ k1 < k2 < n ∧
        k1 < i ⇒ A[k1] ≤ A[k2]) ∧
(∀k. i1 ≤ k < j ∧
        k ≥ 0 ⇒ A[v] ≤ A[k])
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 \( \phi \) (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 ↔ Synthesis Tools

Program optimization
Program sketching
Programming by examples
Invariant generation

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 $\varphi$
    - 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 $\varphi[f/e]$ 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: \((x \leq f(x,y)) \land (y \leq f(x,y)) \land (f(x,y) = x \lor f(x,y) = y)\)

- Candidate Implementations: Linear expressions
  \[ \text{LinExp} := x \mid y \mid \text{Const} \mid \text{LinExp} + \text{LinExp} \mid \text{LinExp} - \text{LinExp} \]

- No solution exists
SyGuS Example

- Theory QF-LIA
- Function to be synthesized: $f \ (\text{int} \ x, \ \text{int} \ y) : \text{int}$
- Specification: $(x \leq f(x,y)) \land (y \leq f(x,y)) \land (f(x,y) = x \ | \ f(x,y) = y)$
- Candidate Implementations: Conditional expressions without +
  
  Term := $x \ | \ y \ | \ \text{Const} \ | \ \text{If-Then-Else} \ (\text{Cond}, \ \text{Term}, \ \text{Term})$
  
  Cond := Term <= Term \ | \ Cond \ & \ Cond \ | \ \sim \ Cond \ | \ (\text{Cond})$

- Possible solution:
  
  If-Then-Else $(x \leq y, \ y, \ x)$
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 \( f(\text{int } x) : \text{int} \)
  \[ x \leq f(x) \& -x \leq f(x) \]

- Set \( E \) of implementations: Conditional linear expressions

- Multiple solutions are possible
  - If-Then-Else \( (0 \leq x, x, 0) \)
  - If-Then-Else \( (0 \leq x, x, -x) \)

- 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

- **Goal**: Find inductive loop invariant automatically

- **Function to be synthesized**
  \[ \text{Inv}(\text{bool } x, \text{bool } z, \text{int } a, \text{int } b) : \text{bool} \]

- **Compile loop-body into a logical predicate**
  \[ \text{Body}(x, y, z, a, b, c, x', y', z', a', b', c') \]

- **Specification**:
  \[ \text{Inv} \land \text{Body} \land \text{Test}' \implies \text{Inv}' \]

- **Template for set of candidate invariants**
  \[
  \text{Term} := a \mid b \mid \text{Const} \mid \text{Term} + \text{Term} \mid \text{If-Then-Else (Cond, Term, Term)} \\
  \text{Cond} := x \mid z \mid \text{Cond} \land \text{Cond} \mid \neg \text{Cond} \mid (\text{Cond})
  \]
Safety Verification of Transition Systems

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

Safety verification problem: Given a property $\varphi(X)$, show that every reachable state of $S$ satisfies $\varphi$

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 $\text{Inv}(X)$ s.t.
1. $\text{Init}(X) \rightarrow \text{Inv}(X)$
2. $\text{Inv}(X) \rightarrow \neg \varphi(X)$
3. $\text{Inv}(X) \& \text{Trans}(X, X') \rightarrow \text{Inv}(X')$
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
  
  \[
  f(A,B)_{[1,1]} = A_{[1,1]}B_{[1,1]} + A_{[1,2]}B_{[2,1]} \
  \ldots
  \]

- 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)
There are five buckets arranged in a circle. Each bucket can hold up to $B$ 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 $\exists a,b,c. \forall X. \phi [ f/ ax+by+c]$
  - 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

Initial examples I

Learning Algorithm

Verification Oracle

Candidate Expression

Counterexample

Fail

Success

Concept class: Set $E$ of expressions

Examples: Concrete input values
Counter-Example Guided Inductive Synthesis

- Concrete inputs $I$ for learning $f(x,y) = \{(x=a,y=b), (x=a',y=b'), \ldots\}$

- Learning algorithm proposes candidate expression $e$ such that $\varphi[f/e]$ holds for all values in $I$

- Check if $\varphi[f/e]$ is valid for all values using SMT solver

- If valid, then stop and return $e$

- If not, let $(x=\alpha, y=\beta, \ldots)$ be a counter-example (satisfies $\sim \varphi[f/e]$)

- Add $(x=\alpha, y=\beta)$ to tests $I$ for next iteration
CEGIS Example

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

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

Examples = \{ \}

Candidate
\[ f(x,y) = x \]

Learning Algorithm

Verification Oracle

Example
\((x=0, y=1)\)
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

Examples = \{ \((x=0, y=1)\) \}

Candidate \(f(x,y) = y\)

Example \((x=1, y=0)\)
CEGIS Example

- Specification: \((x \leq f(x,y)) \land (y \leq f(x,y)) \land (f(x,y) = x \lor f(x,y) = y)\)

- Set E: All expressions built from \(x, y, 0, 1, \text{Comparison, +, If-Then-Else}\)

Examples = 
\[
\{(x=0, y=1) \quad (x=1, y=0) \\
(x=0, y=0) \quad (x=1, y=1)\}
\]

Candidate
ITE \((x \leq y, y, x)\)

Success
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

<table>
<thead>
<tr>
<th>Synthesized Expr.</th>
<th>Counter-Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>x</td>
<td>(x=0, y =1)</td>
</tr>
<tr>
<td>y</td>
<td>(x=1, y =0)</td>
</tr>
<tr>
<td>1</td>
<td>(x=0, y =0)</td>
</tr>
<tr>
<td>x + y</td>
<td>(x=1, y =1)</td>
</tr>
<tr>
<td>ITE(x&lt;y,y,x)</td>
<td>Verified</td>
</tr>
</tbody>
</table>

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 $\phi[f/e]$ 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.**

\[ o_1 = x \]

\[ o_1 = x < x \]
\[ o_2 = \text{ITE}(o_1, y, x) \]

\[ o_1 = y > x \]
\[ o_2 = \text{ITE}(o_1, y, x) \]

**Counter-Example**

\( (x=-1, y=0) \)

\( (x=0, y=-1) \)

**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 \( n \). \( P(e) \propto \text{Score}(e) \) ("Extent to which \( e \) meets the spec \( \varphi \)").

- For a given set \( I \) of concrete inputs, \( \text{Score}(e) = \exp(-0.5 \cdot \text{Wrong}(e)) \), where \( \text{Wrong}(e) = \) No of examples in \( I \) for which \( \neg \varphi \left[ f/e \right] \).

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

- Initial candidate expression $e$ sampled uniformly from $E_n$
- When $\text{Score}(e) = 1$, 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 $\min\{1, \text{Score}(e')/\text{Score}(e)\}$, replace $e$ with $e'$
- Outer loop responsible for updating expression size $n$
Stochastic CEGIS

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

\[ \text{CEXs} = \{(-1,-4), (-1,-3), (-1,-2), (1,1), (1,2)\} \]

\[ e = \text{ITE}(x<0, y, x) \quad p_I(e) = 1/768 \]
\[ e' = \text{ITE}(y<0, y, x) \quad p_C(e') = 1/6 \times 1/48 \]
\[ p_M(e \rightarrow e') = \min(1, \text{Score}(e')/\text{Score}(e)) \]
\[ = \exp(-0.5) \]

\[ e'' = \text{ITE}(x<y, y, x) \quad p_C(e'') = 1/6 \times 1/48 \]
\[ p_M(e \rightarrow e'') = \min(1, \text{Score}(e'')/\text{Score}(e)) \]
\[ = 1 \]
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 $1 - \frac{1}{2^n}$ 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 maintains the following invariant:

\[(a_1 + a_3 < 1) \& (a_2 \leq 1) \& (a_4 = 0) \& (a_5 = 0)\]

\(a_1, a_2, a_3, a_4, a_5\): 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

- **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)**

www.sygus.org
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

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

- Invariants
- Model Check/Verify
- Verified Protocol
- Counter-example
I know exactly what to do in this particular scenario!!

BUT...

Need to figure out how to fix the code
Can we make the process of specifying distributed protocols easier?
Scenarios:
- Describe execution traces
- Translated from informal specs
- Can be symbolic
- Can be concrete
### TRANSIT Specification

<table>
<thead>
<tr>
<th>load</th>
<th>store</th>
<th>replacement</th>
<th>Fwd/Get/S</th>
<th>Fwd/Get/M</th>
<th>Inv</th>
<th>Put-Ack</th>
<th>Data from Dir (ack=0)</th>
<th>Data from Dir (ack&gt;0)</th>
<th>Data from Owner</th>
<th>Inv-Ack</th>
<th>Last-Inv-Ack</th>
</tr>
</thead>
<tbody>
<tr>
<td>I</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ISD</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>IMAD</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>IMA</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>S</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SMAD</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SMA</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>M</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MIA</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SIA</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>IIA</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
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

Directory

Cache1

Cache2

Cache3

Shared

Invalid

Busy

Inv-M

GetM

Acks=1

Inv

Inv

Inv
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:
\[ \text{ReqMsg.Sender} = 1 \land \text{ReqMsg.Type} = \text{GetM} \land \text{Sharers} = \{2,3\} \]

Update:
\[ \text{RespMsg.Acks} = 2; \]
\[ \text{RespMsg.Destination} = 1; \]
\[ \text{InvMsg.Destination} = \{2,3\} \]

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
To generate a completed protocol TRANSIT needs to:

- Find guards and updates consistent with given snippets

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:
\[ \text{ReqMsg.Sender} = 1 \land \text{ReqMsg.Type} = \text{GetM} \land \text{Sharers} = \{1,2\} \]

Update:
\[ \text{RespMsg.Acks} = 1; \]
\[ \text{InvMsg.Destination} = \{2\} \]

Based on two snippets, synthesis tool computes update:

\[ \text{RespMsg.Acks} = \text{Size(Sharers + ReqMsg.Sender)} - 1; \]
\[ \text{InvMsg.Destination} = \text{Sharers} - \text{ReqMsg.Sender} \]
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

<table>
<thead>
<tr>
<th></th>
<th>VI Protocol</th>
<th>MSI Protocol</th>
</tr>
</thead>
<tbody>
<tr>
<td>Snippets used</td>
<td>19</td>
<td>77</td>
</tr>
<tr>
<td>Update expressions synthesized</td>
<td>49</td>
<td>157</td>
</tr>
<tr>
<td>Guard expressions synthesized</td>
<td>17</td>
<td>45</td>
</tr>
<tr>
<td>Expressions explored</td>
<td>3.1K</td>
<td>44.5K</td>
</tr>
<tr>
<td>Synthesis time</td>
<td>5 sec</td>
<td>134 sec</td>
</tr>
<tr>
<td>States explored by Murphi</td>
<td>140K</td>
<td>154K</td>
</tr>
</tbody>
</table>
Methodology Evaluation

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

- 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>
<thead>
<tr>
<th>TABLE 8.2: MSI Directory Protocol—Directory Controller</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>I</strong></td>
</tr>
<tr>
<td><strong>S</strong></td>
</tr>
<tr>
<td><strong>M</strong></td>
</tr>
<tr>
<td><strong>S^D</strong></td>
</tr>
</tbody>
</table>

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 not S already), and adds the requestor to the sharen list. When the Data arrives at the requestor, the
Sample Scenarios

Transition from I to S

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