

LLMs + Program Synthesis

A Journey

Aalok Thakkar

Program Synthesis

Program Synthesis

Declarative Knowledge

$$\forall x \in \mathbb{R}^{\geq 0}. \exists y \in \mathbb{R}^{\geq 0}. (y^2 = x)$$

Imperative Knowledge

$$y_{n+1} = \frac{y_n^2 + x}{2y_n}$$

Program Synthesis

Declarative Knowledge

$$\forall x \in \mathbb{R}^{\geq 0}. \exists y \in \mathbb{R}^{\geq 0}. (y^2 = x)$$

User Intent

Imperative Knowledge

$$y_{n+1} = \frac{y_n^2 + x}{2y_n}$$

Implementation

Synthesis: Dreams \Rightarrow Programs

Zohar Manna and Richard Waldinger

IEEE Transactions on Software Engineering, 1979

Can Programming Be Liberated, Period?

David Harel

IEEE, 2008

Can Programming Be Liberated, Period?

David Harel

IEEE, 2008

More Computing Power

Mature Software Analysis and Verification Tools

Better Human-Computer Interfaces

Data Mining tools for Code Repositories

Programming by Example

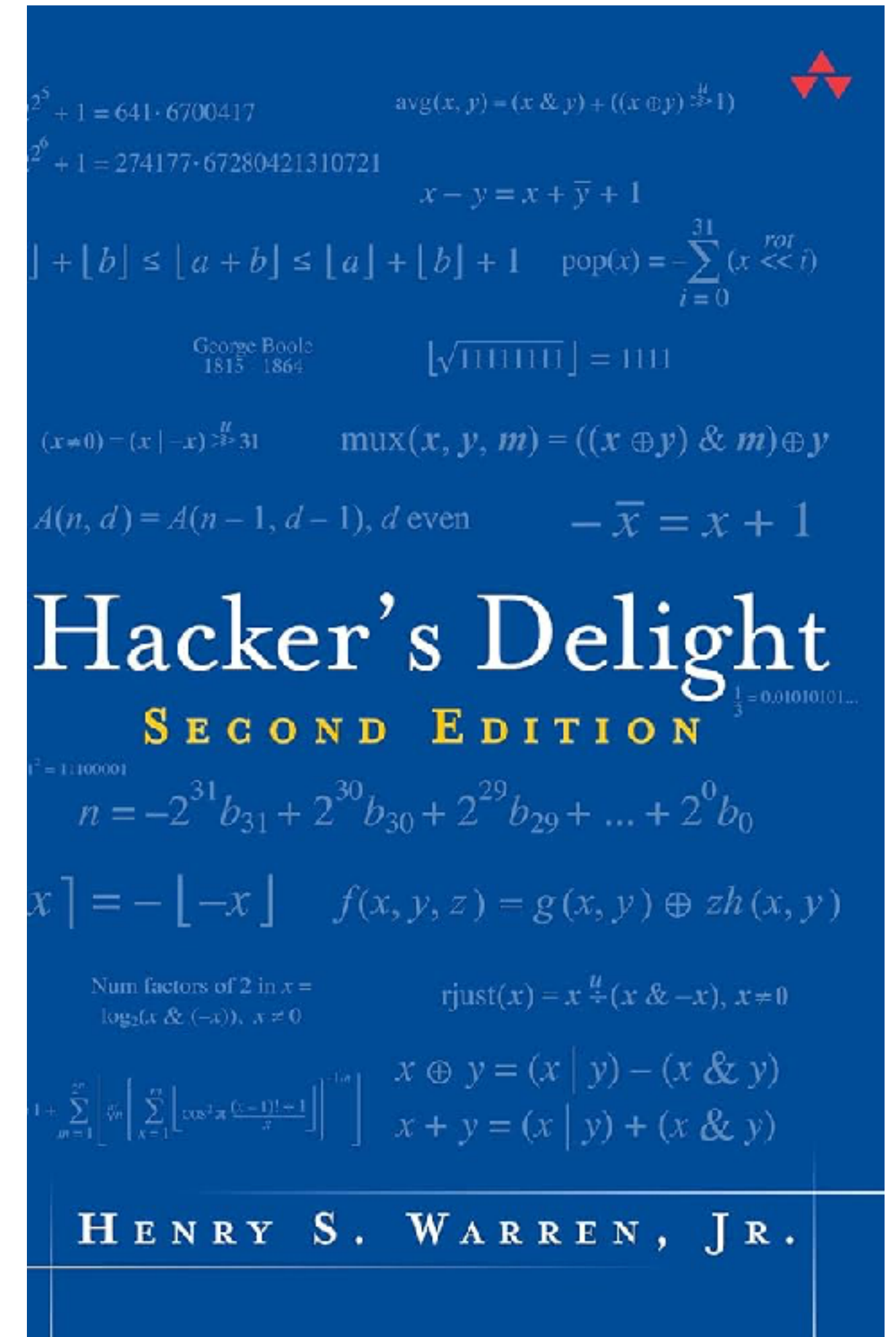
Desired program \mathbb{P} : A bit-vector transformation that resets the rightmost substring of contiguous 1s to 0s

1. \mathbb{P} should be constructed from standard bit-vector operations
 $|, \&, \sim, +, -, \ll, \gg, 0, 1 \dots$
2. \mathbb{P} can be specified using examples:

00101 \rightarrow 00100

01010 \rightarrow 01000

10110 \rightarrow 10000



Programming by Example

Desired program \mathbb{P} : A bit-vector transformation that resets the rightmost substring of contiguous 1s to 0s

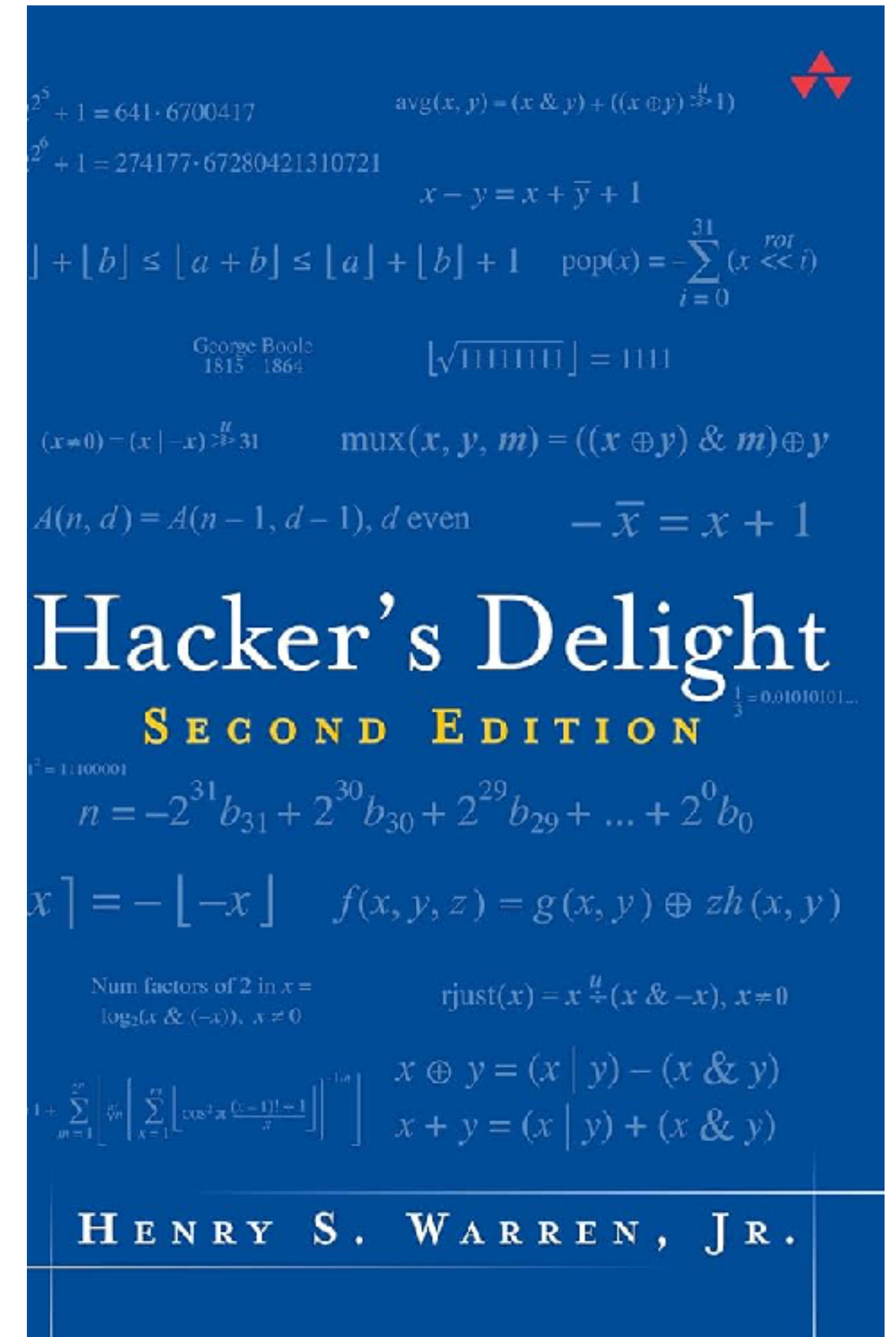
1. \mathbb{P} should be constructed from standard bit-vector operations
 $|, \&, \sim, +, -, \ll, \gg, 0, 1 \dots$
2. \mathbb{P} can be specified using examples:

00101 \rightarrow 00100

01010 \rightarrow 01000

10110 \rightarrow 10000

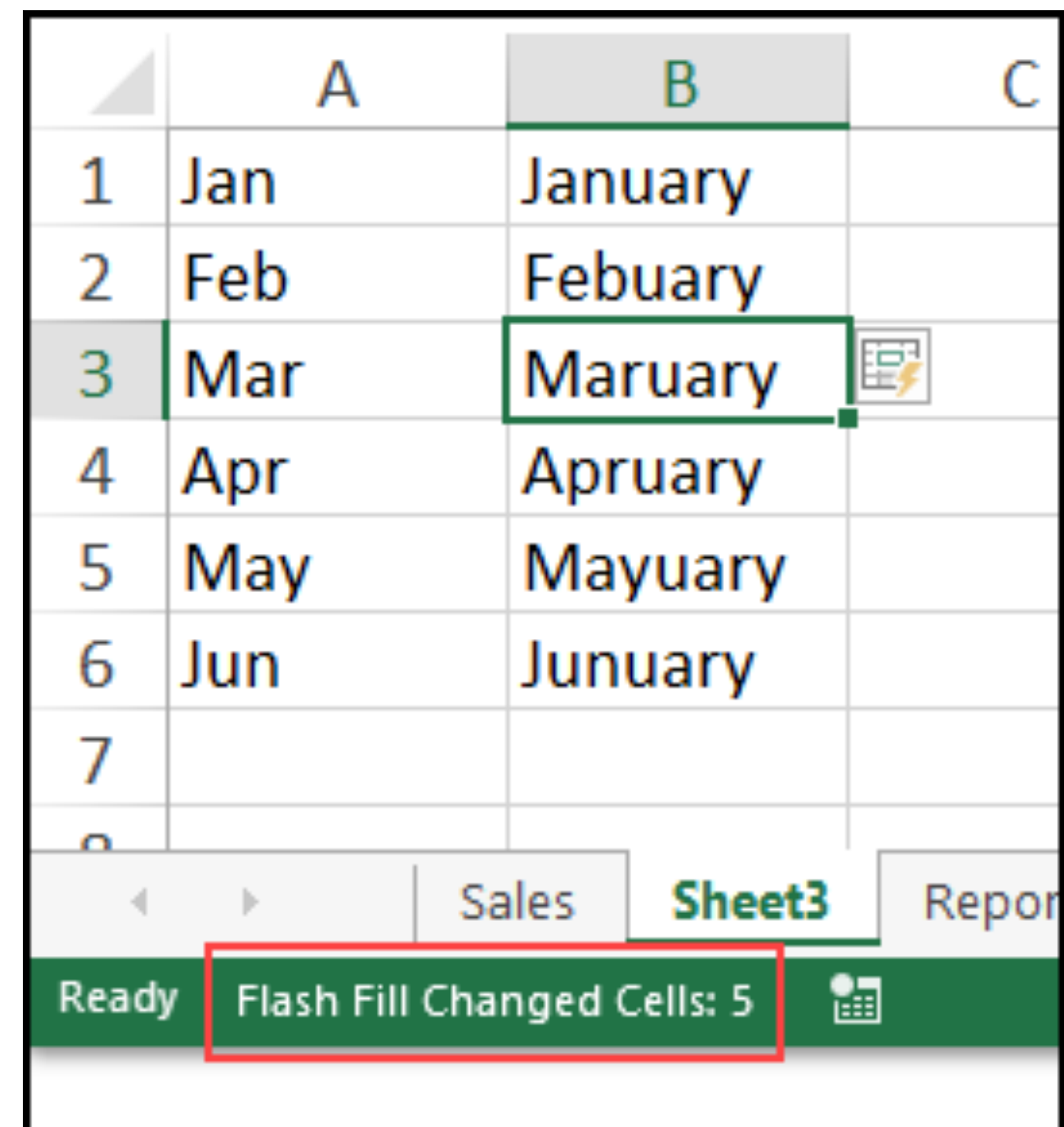
$x \& (1 + (x | (x - 1)))$



Automating String Processing in Spreadsheets Using Input-Output Examples

Sumit Gulwani

POPL 2011



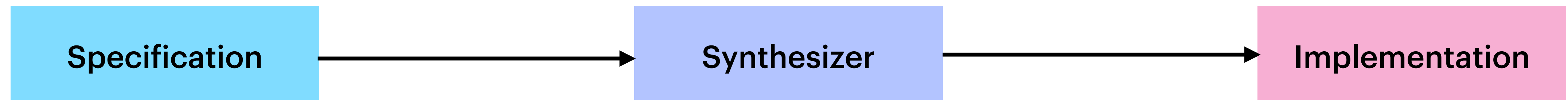
| | A | B | C |
|---|-----|----------|---|
| 1 | Jan | January | |
| 2 | Feb | Febuary | |
| 3 | Mar | Maruuary | |
| 4 | Apr | Apruary | |
| 5 | May | Mayuary | |
| 6 | Jun | Junuary | |
| 7 | | | |
| 8 | | | |

Ready Flash Fill Changed Cells: 5

Syntax-Guided Synthesis

Rajeev Alur, Rastislav Bodik, Garvit Juniwal, Milo M. K. Martin, Mukund Raghothaman,
Sanjit A. Seshia, Rishabh Singh, Armando Solar-Lezama, Emina Torlak, Abhishek Udupa

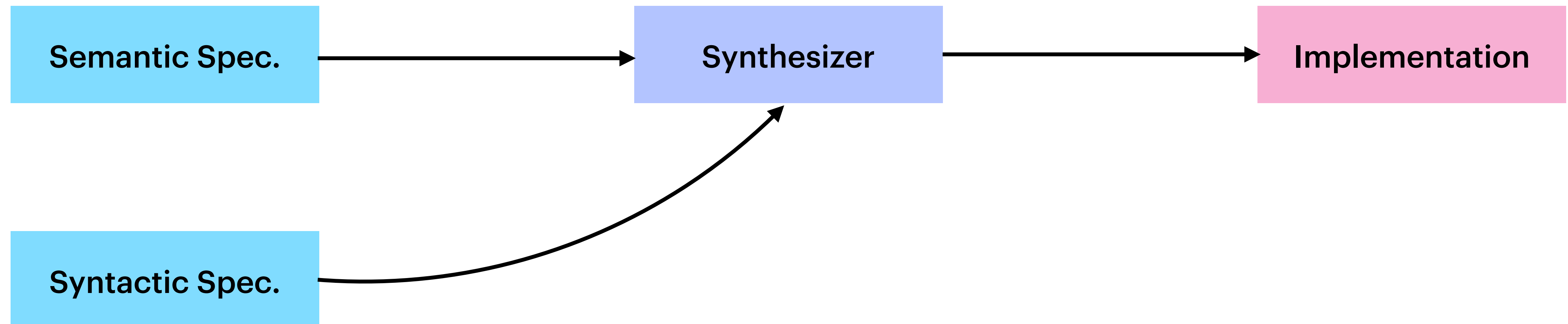
FMCAD 2013



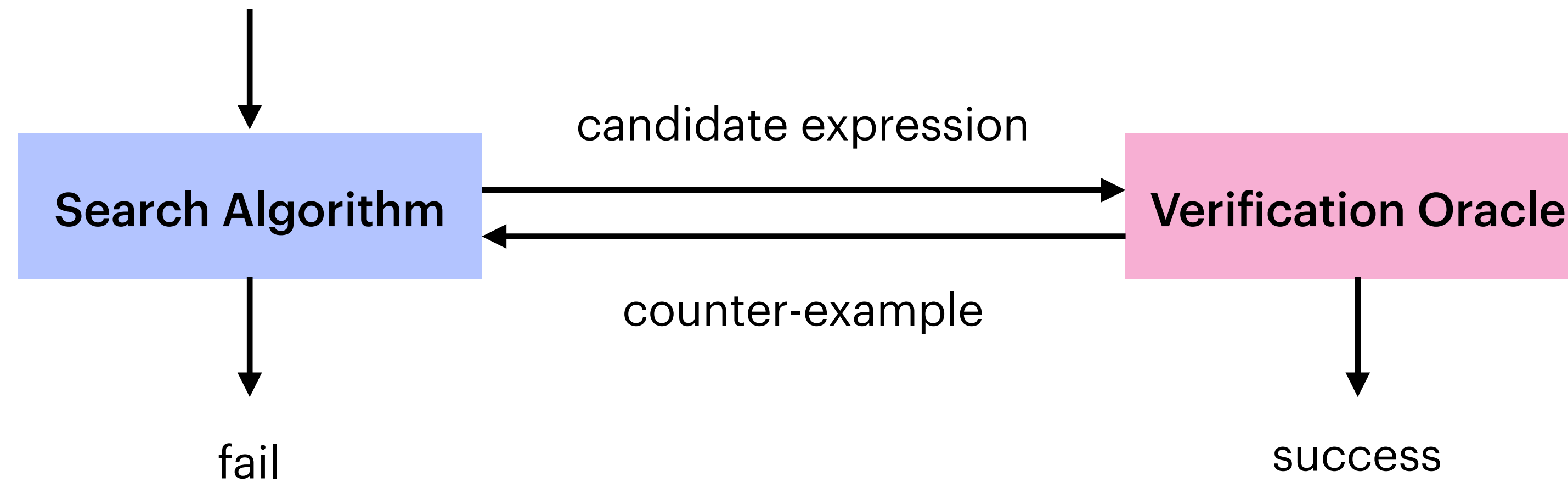
Syntax-Guided Synthesis

Rajeev Alur, Rastislav Bodik, Garvit Juniwal, Milo M. K. Martin, Mukund Raghothaman,
Sanjit A. Seshia, Rishabh Singh, Armando Solar-Lezama, Emina Torlak, Abhishek Udupa

FMCAD 2013



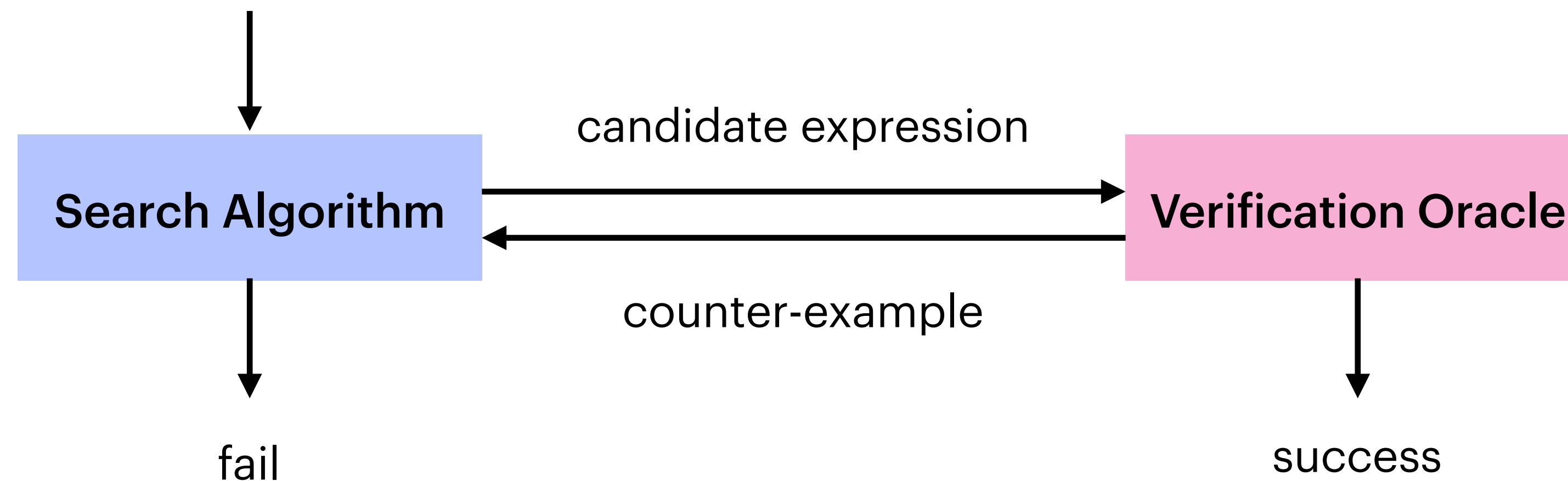
Syntax-Guided Synthesis



Combinatorial Sketching for Finite Programs

Armando Solar-Lezama, Liviu Tancau, Rastislav Bodik, Vijay Saraswat, Sanjit Seshia

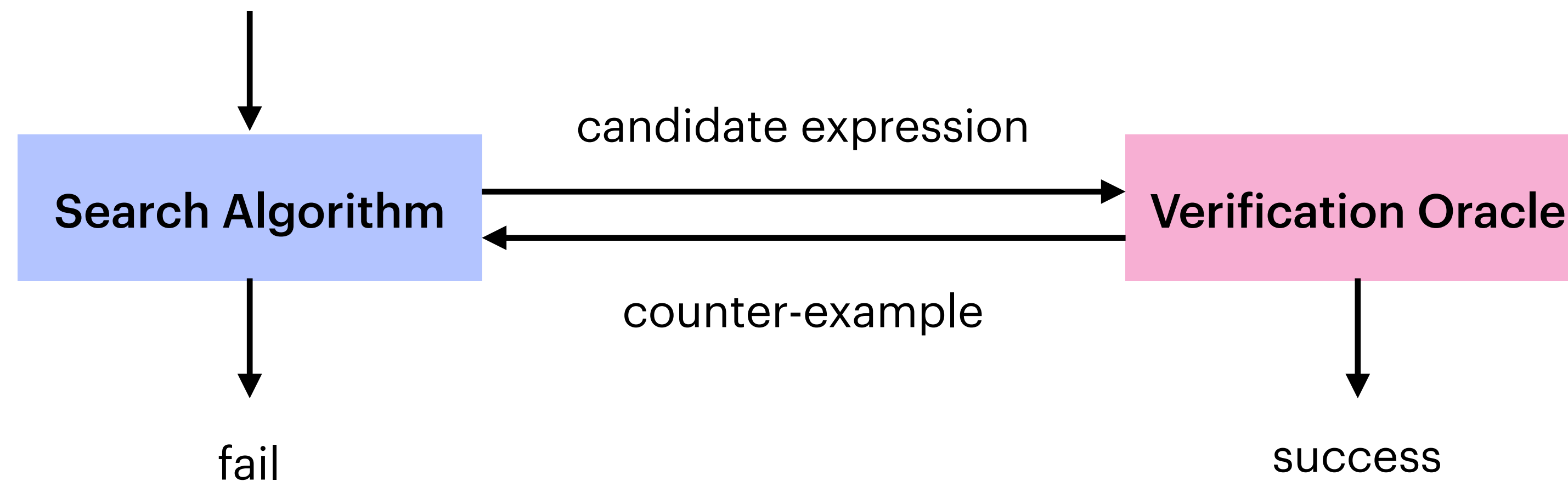
ASPLOS 2006



Combinatorial Sketching for Finite Programs

Armando Solar-Lezama, Liviu Tancau, Rastislav Bodik, Vijay Saraswat, Sanjit Seshia

ASPLOS 2006



$$(x \leq f(x, y)) \wedge (y \leq f(x, y)) \wedge (f(x, y) = x \vee f(x, y) = y)$$

Initial Learning Strategies:

Enumerative Search (searching with pruning): Udupa et. al. (PLDI 2013)

Symbolic Search (solving constraints): Gulwani et. al. (PLDI 2011)

Stochastic (probablistic walk): Schkufza et. al. (ASPLOS 2013)

Initial Learning Strategies:

Enumerative Search (searching with pruning): Udupa et. al. (PLDI 2013)

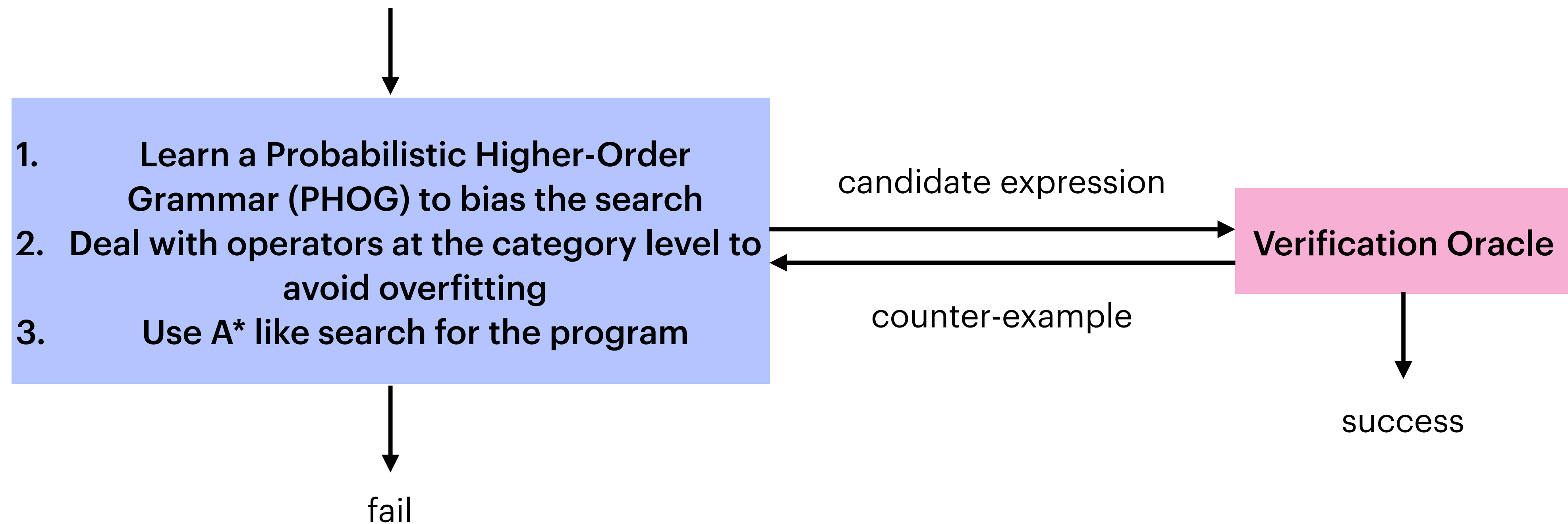
Symbolic Search (solving constraints): Gulwani et. al. (PLDI 2011)

Stochastic (probablistic walk): Schkufza et. al. (ASPLOS 2013)

Accelerating Search-Based Program Synthesis using Learned Probabilistic Models

Woosuk Lee, Kihong Heo, Rajeev Alur, Mayur Naik

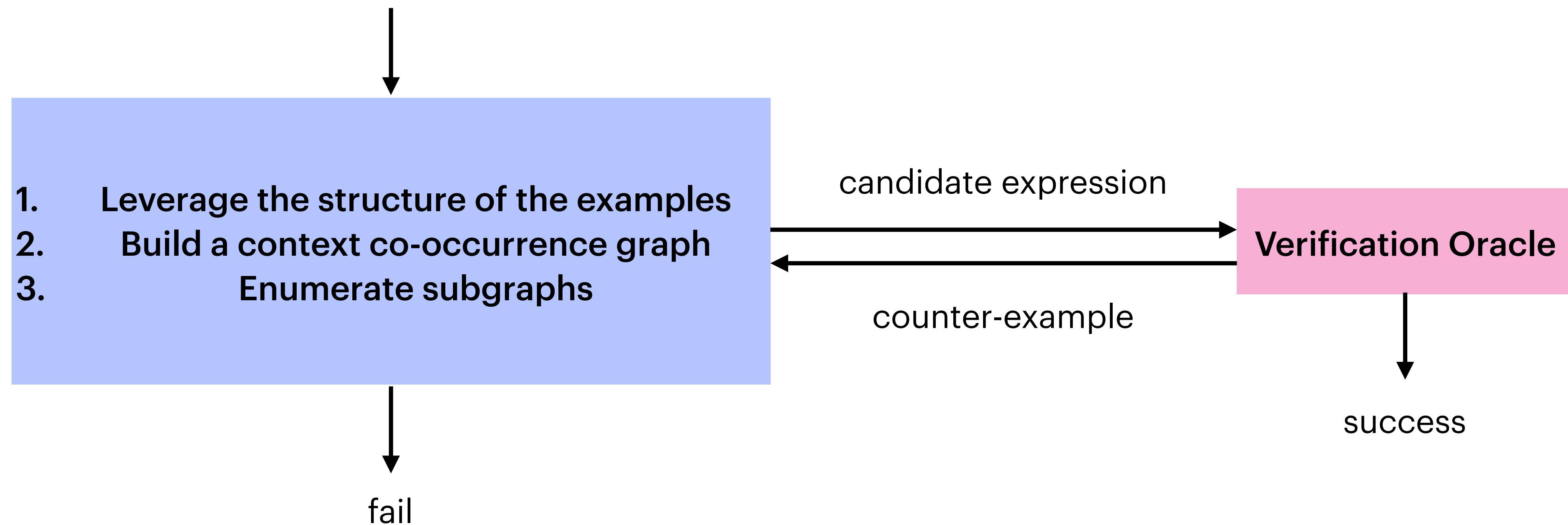
PLDI 2018



Example-Guided Synthesis of Relational Queries

Aalok Thakkar, Aaditya Naik, Nathaniel Sands, Rajeev Alur,
Mayur Naik, Mukund Raghothaman

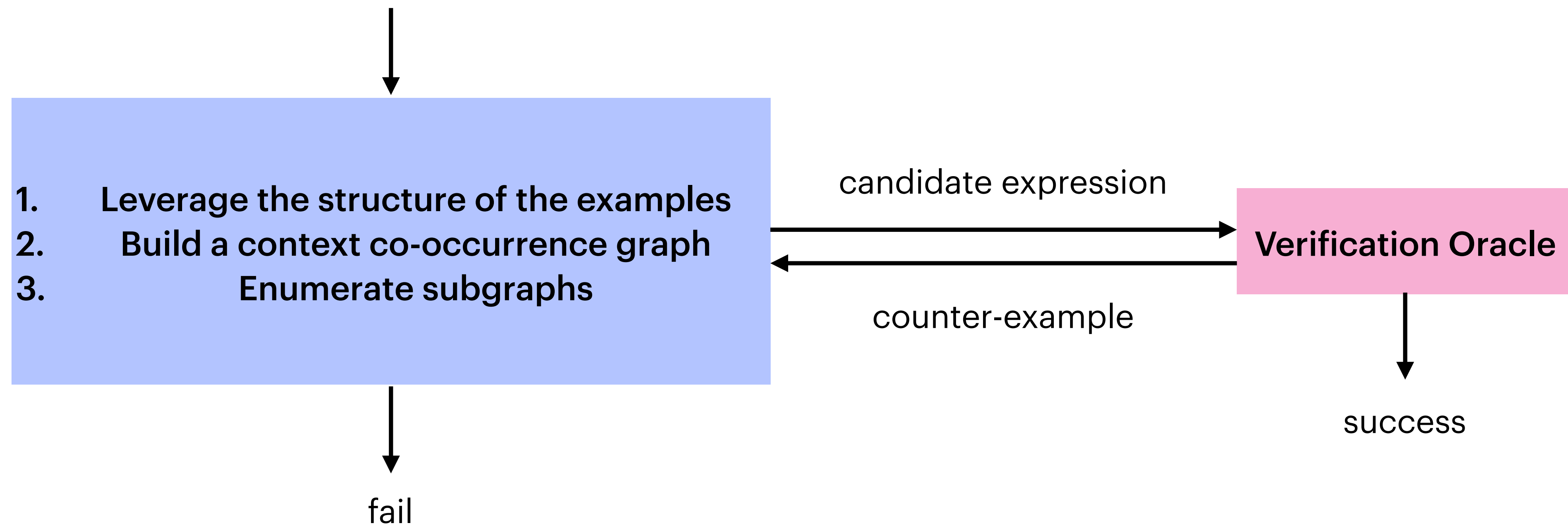
PLDI 2021



Example-Guided Synthesis of Relational Queries

Aalok Thakkar, Aaditya Naik, Nathaniel Sands, Rajeev Alur,
Mayur Naik, Mukund Raghothaman

PLDI 2021



can LLMs solve all of these?

can LLMs solve all of these?

Find a function $f(x, y)$ using only symbols $x, y, +, -, =, <, \text{if-then-else}$, and boolean operators $\text{and}, \text{or}, \text{and not}$ such that $(x \leq f(x, y))$ and $(y \leq f(x, y))$ and $(f(x, y) = x \text{ or } f(x, y) = y)$.



We are given the task of defining a function $f(x, y)$ using **only the symbols**:

- **Variables:** x, y
- **Operators/syntax:** $+, -, =, <$
- **Control/logic:** `if-then-else`, `and`, `or`, `not`

We are to construct $f(x, y)$ such that it satisfies **three conditions**:

1. $x \leq f(x, y)$
2. $y \leq f(x, y)$
3. $f(x, y) = x \text{ or } f(x, y) = y$

Step-by-step Analysis:

Conditions (1) and (2) together imply:

$$\begin{aligned} f(x, y) &\geq x \text{ and } f(x, y) \geq y \\ \rightarrow f(x, y) &\geq \max(x, y) \end{aligned}$$

But condition (3) tells us:

$$f(x, y) \in \{x, y\}$$

So for all three conditions to hold, $f(x, y)$ **must be the maximum of x and y** :

$$f(x, y) = \max(x, y)$$

Expressing `max(x, y)` with allowed symbols

We can write this using `if-then-else` logic:

$$f(x, y) = \text{if } x < y \text{ then } y \text{ else } x$$

Jigsaw: Large Language Models meet Program Synthesis

Naman Jain, Skanda Vaidyanath, Arun Iyer, Nagarajan Natarajan, Suresh Parthasarathy,
Sriram Rajamani, Rahul Sharma

ICSE 2022

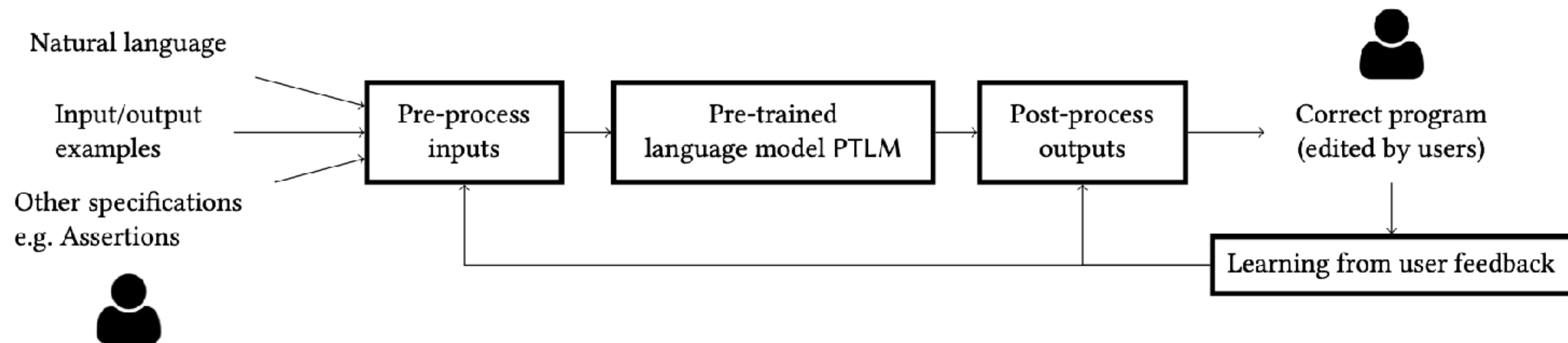


Figure 2: Architecture of Jigsaw

Jigsaw: Large Language Models meet Program Synthesis

Naman Jain, Skanda Vaidyanath, Arun Iyer, Nagarajan Natarajan, Suresh Parthasarathy,
Sriram Rajamani, Rahul Sharma

ICSE 2022

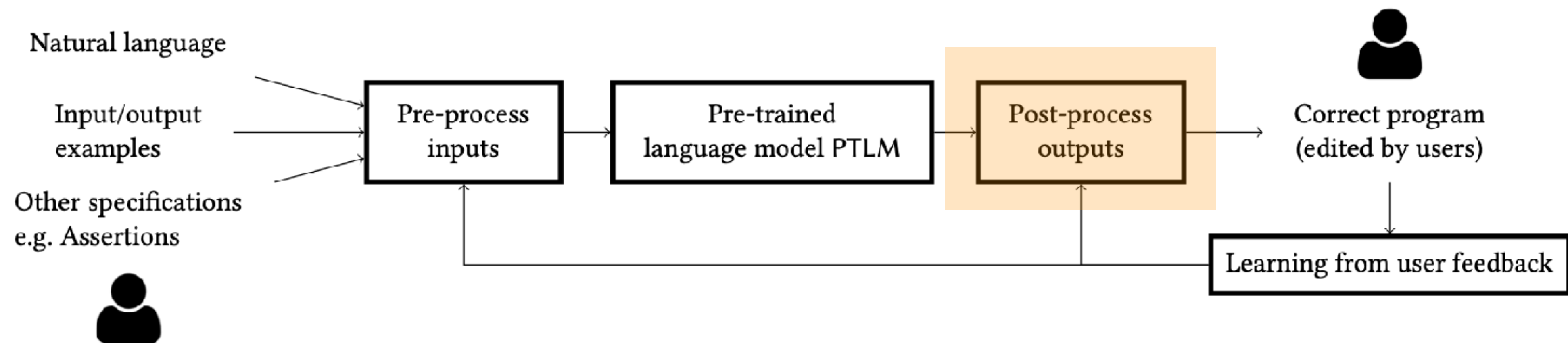


Figure 2: Architecture of Jigsaw

Jigsaw: Large Language Models meet Program Synthesis

Naman Jain, Skanda Vaidyanath, Arun Iyer, Nagarajan Natarajan, Suresh Parthasarathy,
Sriram Rajamani, Rahul Sharma

ICSE 2022

1. Correctness on IO Examples
2. Variable Name Transformations
3. Argument Transformations
4. AST-to-AST Transformations
5. Learning from User Feedback

Guiding Enumerative Program Synthesis with Large Language Models

Yixuan Li, Julian Parsert, and Elizabeth Polgreen

CAV 2024

Guiding Enumerative Program Synthesis with Large Language Models

Step 1: Can off-the-shelf LLMs solve program synthesis?

Guiding Enumerative Program Synthesis with Large Language Models

Step 1: Can off-the-shelf LLMs solve program synthesis?

They craft a library of prompts that solve roughly 50% of the SyGuS benchmarks.

Guiding Enumerative Program Synthesis with Large Language Models

Step 1: Can off-the-shelf LLMs solve program synthesis?

They craft a library of prompts that solve roughly 50% of the SyGuS benchmarks.

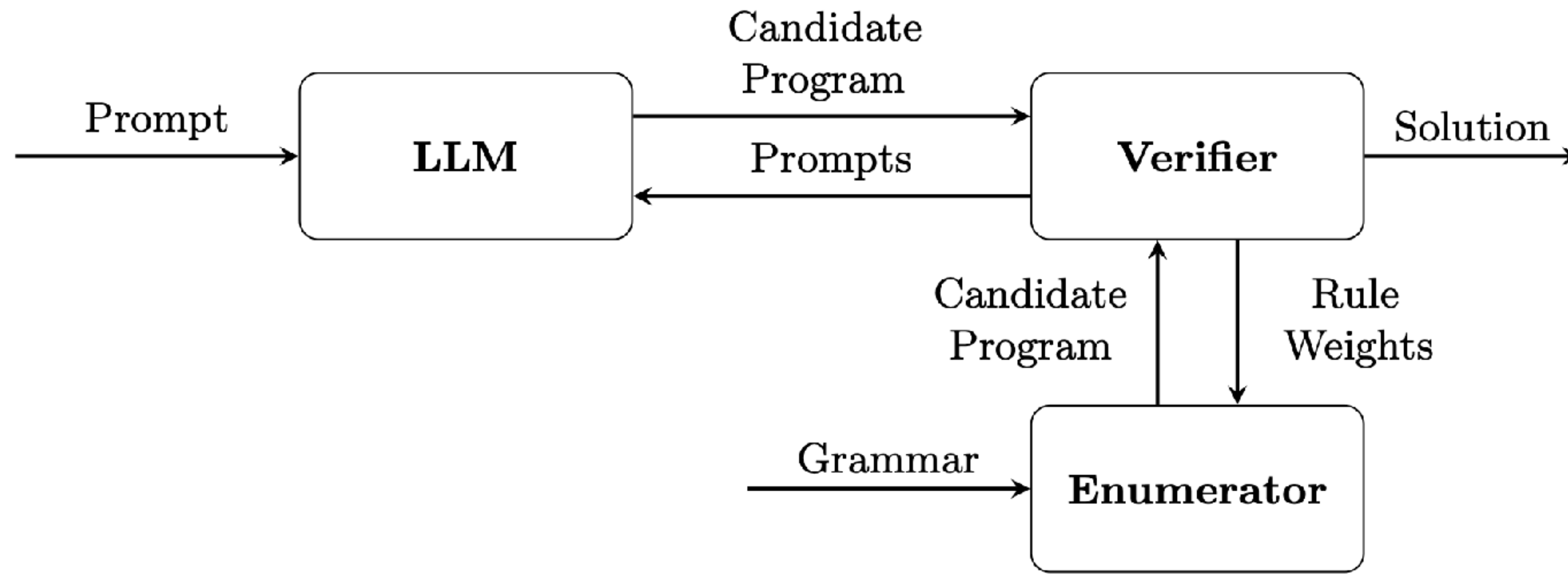
In the cases where the LLM returns incorrect solutions, the correct solutions are most often in the vicinity of the incorrect solutions.

Guiding Enumerative Program Synthesis with Large Language Models

Step 2: Bias counter-example guided inductive synthesis

Use the incorrect programs to create a probabilistic CFG and use it to guide an enumerative synthesizer within a CEGIS loop.

Guiding Enumerative Program Synthesis with Large Language Models



Stand-alone LLM

1. Use off-the-shelf LLM (GPT-3.5-turbo) without fine tuning
2. Rename any functions and variables to avoid generation based on name
3. Ask LLM to generate first in Lisp (6% better) or Python than SMT-LIB
4. Use emotional prompts (8% better)
5. Then convert from Lisp (or Python) to SMT-LIB

Stand-alone LLM

1. Use off-the-shelf LLM (GPT-3.5-turbo) without fine tuning
2. Rename any functions and variables to avoid generation based on name
3. Ask LLM to generate first in Lisp (6% better) or Python than SMT-LIB
4. Use emotional prompts (8% better)
5. Then convert from Lisp (or Python) to SMT-LIB

```
(set-logic LIA)
(synth-fun fn ((vr0 Int) (vr1 Int) (vr2 Int)) Int)
(constraint (>= (fn vr0 vr1 vr2) vr0))
(constraint (>= (fn vr0 vr1 vr2) vr1))
(constraint (>= (fn vr0 vr1 vr2) vr2))
(constraint (or (= vr0 (fn vr0 vr1 vr2)) (or (= vr1 (fn vr0 vr1 vr2)) (=
    vr2 (fn vr0 vr1 vr2)))))
(check-synth)
```

```
(set-logic LIA)
(declare-var vr0 Int)
(declare-var vr1 Int)
(declare-var vr2 Int)
(constraint (>= (fn vr0 vr1 vr2) vr0))
(constraint (>= (fn vr0 vr1 vr2) vr1))
(constraint (>= (fn vr0 vr1 vr2) vr2))
(constraint (or (= vr0 (fn vr0 vr1 vr2)) (or (= vr1 (fn vr0 vr1 vr2)) (=
    vr2 (fn vr0 vr1 vr2)))))
(check-synth)
```

You are a good synthesizer. Do you know what "(define-fun fn ((vr0 Int) (vr1 Int) (vr2 Int)) Int)" is doing?

Write only one Lisp-like method "defun fn" without any built-in methods or arrays.

Requirements:

1. No built-in functions.
2. Never violate the SMT-LIB constraints above.
3. Pay attention to the define functions.
4. Ensure the response contains one and only one function.
5. Do not include any iterations, BitVec, or Int notations in the function body.

Write it correctly, or I will lose my job and 100 grandmothers will die.

Don't disappoint me.

Write only one Lisp-like method "defun fn" that never violates the SMT-LIB constraints above.

```
(set-logic LIA)
(declare-var vr0 Int)
(declare-var vr1 Int)
(declare-var vr2 Int)
(constraint (>= (fn vr0 vr1 vr2) vr0))
(constraint (>= (fn vr0 vr1 vr2) vr1))
(constraint (>= (fn vr0 vr1 vr2) vr2))
(constraint (or (= vr0 (fn vr0 vr1 vr2)) (or (= vr1 (fn vr0 vr1 vr2)) (=
    vr2 (fn vr0 vr1 vr2)))))
(check-synth)
```

You are a good synthesizer. Do you know what "(define-fn fn ((vr0 Int) (vr1 Int) (vr2 Int)) Int)" is doing?

Write only one Lisp-like method "defun fn" without any built-in methods or arrays.

Requirements:

1. No built-in functions.
2. Never violate the SMT-LIB constraints above.
3. Pay attention to the define functions.
4. Ensure the response contains one and only one function.
5. Do not include any iterations, BitVec, or Int notations in the function body.

Write it correctly, or I will lose my job and 100 grandmothers will die.

Don't disappoint me.

Write only one Lisp-like method "defun fn" that never violates the SMT-LIB constraints above.


```
(set-logic LIA)
(declare-var vr0 Int)
(declare-var vr1 Int)
(declare-var vr2 Int)
(constraint (>= (fn vr0 vr1 vr2) vr0))
(constraint (>= (fn vr0 vr1 vr2) vr1))
(constraint (>= (fn vr0 vr1 vr2) vr2))
(constraint (or (= vr0 (fn vr0 vr1 vr2)) (or (= vr1 (fn vr0 vr1 vr2)) (=
    vr2 (fn vr0 vr1 vr2)))))
(check-synth)
```

You are a good synthesizer. Do you know what "(define-fun fn ((vr0 Int) (vr1 Int) (vr2 Int)) Int)" is doing?

Write only one Lisp-like method "defun fn" without any built-in methods or arrays.

Requirements:

1. No built-in functions.
2. Never violate the SMT-LIB constraints above.
3. Pay attention to the define functions.
4. Ensure the response contains one and only one function.
5. Do not include any iterations, BitVec, or Int notations in the function body.

**Write it correctly, or I will lose my job and 100 grandmothers will die.
Don't disappoint me.**

Write only one Lisp-like method "defun fn" that never violates the SMT-LIB constraints above.

```
(defun fn (vr0 vr1 vr2)
  (ite (>= vr0 vr1) (ite (>= vr0 vr2) vr0 vr2) (ite (>= vr1 vr2) vr1 vr2)
  )))
```

$f(x, y, z) = \text{if } (x \geq y) \text{ then } (\text{if } (x \geq z) \text{ then } x \text{ else } z) \text{ then } ((y \geq z) \text{ then } y \text{ else } z)$

You are a good programming language converter. Convert the Lisp function to SMT-LIB:

Based on the Lisp code provided above, convert the 'defun' Lisp-like code to a corresponding SMT-LIB function. Use SMT-LIB syntax starting with (define-fun

Follow these guidelines:

1. Only give me the function definition starting with '(define-fun'.
2. Pay attention to types. If there are bit-vector terms, they need to be of the same width.
3. Ensure the SMT-LIB function contains one and only one function definition starting with '(define-fun'.
4. Do not include any iterations, BitVec, or Int notations in the function body.
5. Use the assigned values from the Lisp code during translation.
6. Do not introduce any variables that do not exist in the Lisp function.

Rules for SMT-LIB: +, -, *, ite, >, =, <, >=, <=, and, or, not, true, false.

Results

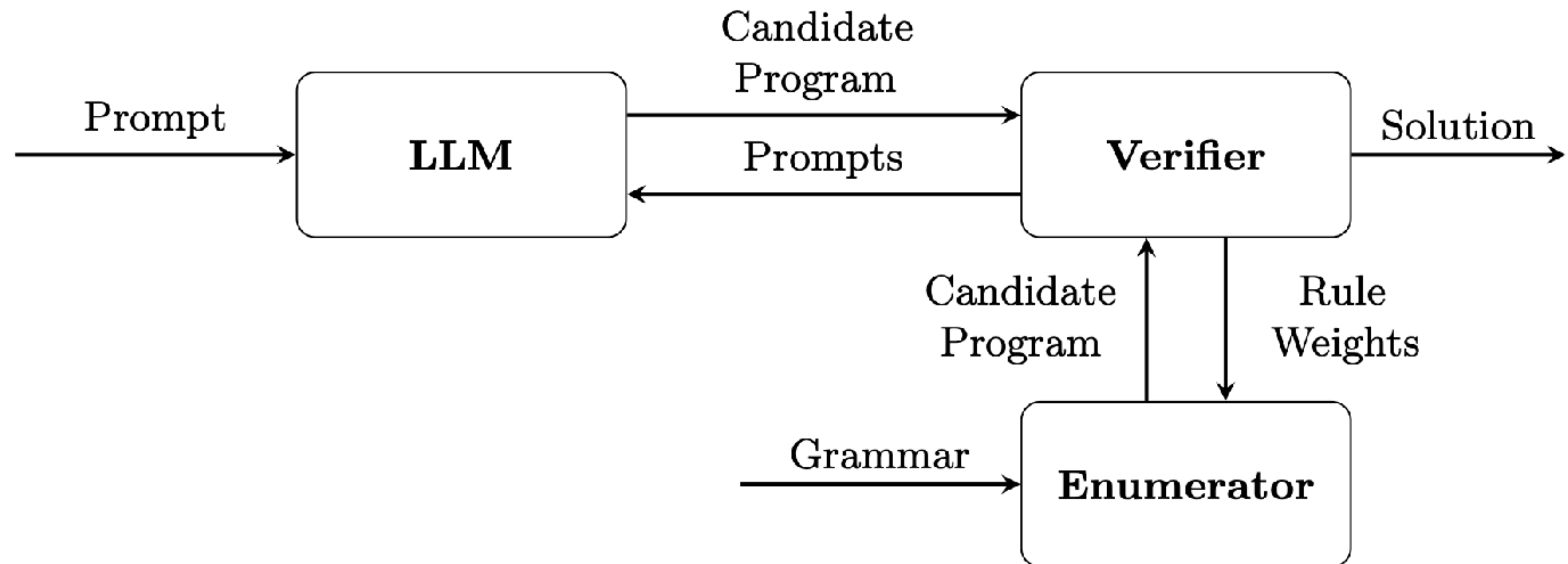
| Method | BV (384) | LIA (87) | INV (138) | Total (609) |
|------------|----------|----------|-----------|---------------|
| Enumerator | 142.7 | 25.0 | 21.0 | 188.7 (31.0%) |
| CVC5 | 292.0 | 43.0 | 80.0 | 415.0 (68.1%) |
| LLM Only | 137.0 | 54.0 | 112.0 | 303.0 (49.8%) |

If the LLM does not generate the correct program...

We want to infer a Weighted CFG that characterises the neighbourhood of the incorrect program.

If the LLM does not generate the correct program...

We want to infer a Weighted CFG that characterises the neighbourhood of the incorrect program.



If the LLM does not generate the correct program...

We want to infer a Weighted CFG that characterises the neighbourhood of the incorrect program.

$$w[r_i] = \sum_p count_r(p)$$

If the LLM does not generate the correct program...

We want to infer a Weighted CFG that characterises the neighbourhood of the incorrect program.

$$w[r_i] = \sum_p count_r(p)$$

$r_1 : \text{Start} \rightarrow (\text{ite StartBool Start Start})$

$r_2 : \text{Start} \rightarrow \text{vr0}$

$r_3 : \text{Start} \rightarrow \text{vr1}$

$r_4 : \text{Start} \rightarrow \text{vr2}$

$r_5 : \text{StartBool} \rightarrow (>= \text{Start Start}).$

```
(defun fn (vr0 vr1 vr2)
  (ite (>= vr0 vr1) (ite (>= vr0 vr2) vr0 vr2) (ite (>= vr1 vr2) vr1 vr2)
  )))
```

$r_1 : \text{Start} \rightarrow (\text{ite StartBool Start Start})$

$r_2 : \text{Start} \rightarrow \text{vr0}$

$r_3 : \text{Start} \rightarrow \text{vr1}$

$r_4 : \text{Start} \rightarrow \text{vr2}$

$r_5 : \text{StartBool} \rightarrow (>= \text{Start Start}).$

```
(defun fn (vr0 vr1 vr2)
  (ite (>= vr0 vr1) (ite (>= vr0 vr2) vr0 vr2) (ite (>= vr1 vr2) vr1 vr2)
  )))
```

$$w[r_1] = 3 \quad w[r_2] = 3 \quad w[r_3] = 3 \quad w[r_4] = 4 \quad w[r_5] = 3$$

$r_1 : \text{Start} \rightarrow (\text{ite StartBool Start Start})$

$r_2 : \text{Start} \rightarrow \text{vr0}$

$r_3 : \text{Start} \rightarrow \text{vr1}$

$r_4 : \text{Start} \rightarrow \text{vr2}$

$r_5 : \text{StartBool} \rightarrow (>= \text{Start Start}).$

```
(defun fn (vr0 vr1 vr2)
  (ite (>= vr0 vr1) (ite (>= vr0 vr2) vr0 vr2) (ite (>= vr1 vr2) vr1 vr2)
  )))
```

$$P(r_1) = \frac{3}{11} \quad P(r_2) = \frac{3}{11} \quad P(r_3) = \frac{3}{11} \quad P(r_4) = \frac{4}{11} \quad P(r_5) = 1$$

$r_1 : \text{Start} \rightarrow (\text{ite StartBool Start Start})$

$r_2 : \text{Start} \rightarrow \text{vr0}$

$r_3 : \text{Start} \rightarrow \text{vr1}$

$r_4 : \text{Start} \rightarrow \text{vr2}$

$r_5 : \text{StartBool} \rightarrow (>= \text{Start Start}).$

Method 1: Top-down Search

+ Hueristics for Pruning

$$P(r_1) = \frac{3}{11} \quad P(r_2) = \frac{3}{11} \quad P(r_3) = \frac{3}{11} \quad P(r_4) = \frac{4}{11} \quad P(r_5) = 1$$

$r_1 : \text{Start} \rightarrow (\text{ite StartBool Start Start})$

$r_2 : \text{Start} \rightarrow \text{vr0}$

$r_3 : \text{Start} \rightarrow \text{vr1}$

$r_4 : \text{Start} \rightarrow \text{vr2}$

$r_5 : \text{StartBool} \rightarrow (>= \text{Start Start}).$

Results

| Method | BV (384) | LIA (87) | INV (138) | Total (609) |
|-------------------------|----------|----------|-----------|---------------|
| CVC5 | 292.0 | 43.0 | 80.0 | 415.0 (68.1%) |
| LLM Only | 137.0 | 54.0 | 112.0 | 303.0 (49.8%) |
| e-pCFG-synth | 196.0 | 24.0 | 100.5 | 245.4 (40.3%) |
| LLM \cup e-pCFG-synth | 255.0 | 64.0 | 117.7 | 436.7 (71.7%) |

Method 2: Weighted A* Search

Based on W. Lee et. al. (PLDI 2018)

Consider $c(x)$ which computes the cost of the path so far, and $g(x)$ which estimates the cost to extend the path to a goal node.

$$c(x) = \sum_{r_i \in D_x} -\log_2 (\mathbb{P}[r_i]) , \quad g(x) = \begin{cases} 0 & \text{if } x \in \Sigma^*, \\ -\sum_{x_i \in V} \log_2 h(x_i) & \text{otherwise,} \end{cases}$$

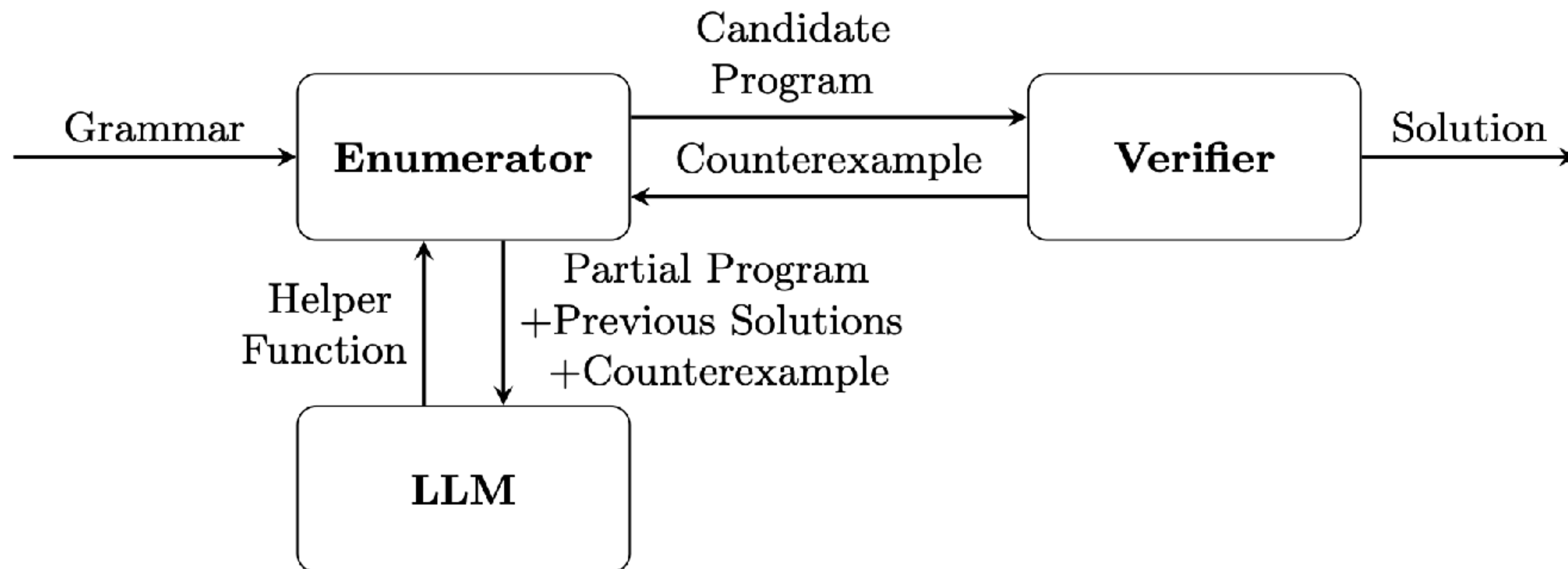
Minimize $c(x) + g(x)$

Results

| Method | BV (384) | LIA (87) | INV (138) | Total (609) |
|--------------------------|----------|----------|-----------|---------------|
| CVC5 | 292.0 | 43.0 | 80.0 | 415.0 (68.1%) |
| LLM Only | 137.0 | 54.0 | 112.0 | 303.0 (49.8%) |
| e-pCFG-synth | 196.0 | 24.0 | 100.5 | 245.4 (40.3%) |
| LLM \cup e-pCFG-synth | 255.0 | 64.0 | 117.7 | 436.7 (71.7%) |
| A* with pCFG-synth | 262.0 | 35.0 | 25.0 | 322 (52.9%) |
| LLM \cup A*-pCFG-synth | 305.0 | 65.0 | 118.0 | 488.0 (80.1%) |

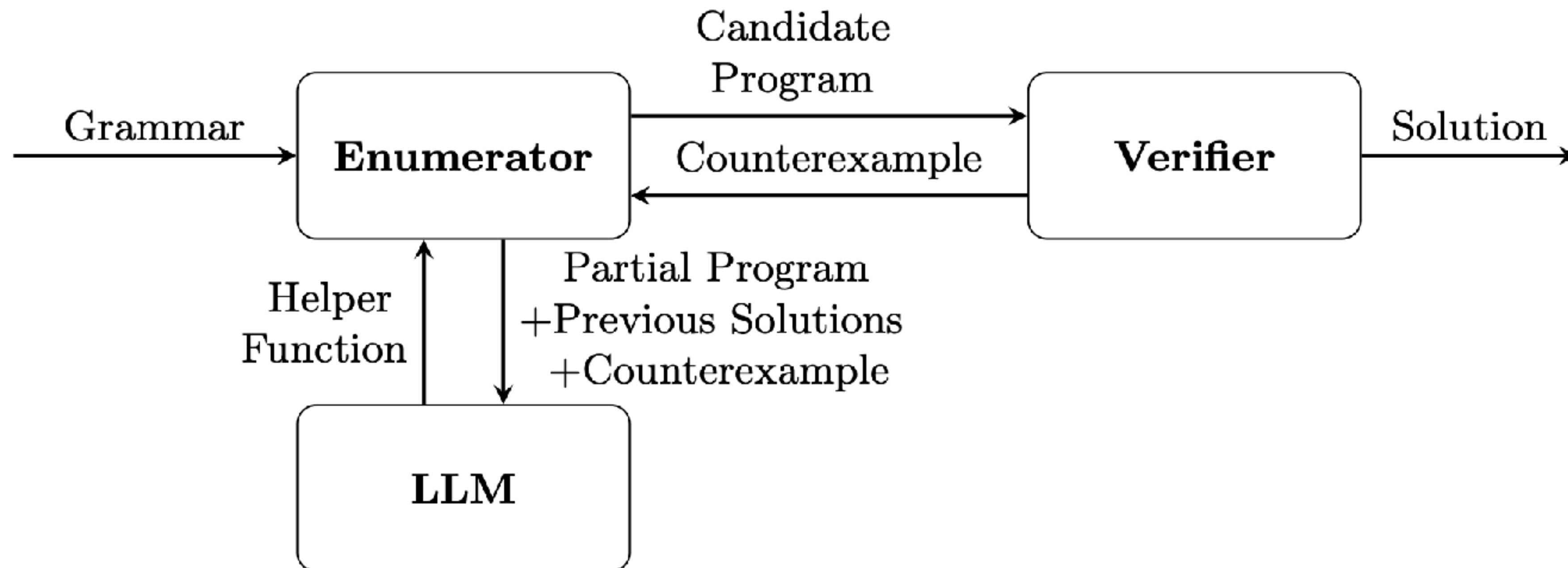
average of 3 runs; timeout 600s

Guiding Enumerative Program Synthesis with Large Language Models



Guiding Enumerative Program Synthesis with Large Language Models

Motivated by Solar-Lezama et. al. (ASPLOS 2006)



You are teaching a student to write SMT-LIB. The student must write a function that satisfies the following constraints:

```
(constraint (>= (fn vr0 vr1 vr2) vr0))  
(constraint (>= (fn vr0 vr1 vr2) vr1))  
(constraint (>= (fn vr0 vr1 vr2) vr2))  
(constraint (or (= vr0 (fn vr0 vr1 vr2)) (or (= vr1 (fn vr0 vr1 vr2)) (= vr2 (fn vr0 vr1 vr2)))))
```

So far, the student has written this code:

```
(define-fun fn ((vr0 Int) (vr1 Int) (vr2 Int)) Int  
  (ite ?? ?? ??))
```

Can you suggest some helper functions for the student to use to complete this code and replace the ??

You are teaching a student to write SMT-LIB. The student must write a function that satisfies the following constraints:

```
(constraint (>= (fn vr0 vr1 vr2) vr0))  
(constraint (>= (fn vr0 vr1 vr2) vr1))  
(constraint (>= (fn vr0 vr1 vr2) vr2))  
(constraint (or (= vr0 (fn vr0 vr1 vr2)) (or (= vr1 (fn vr0 vr1 vr2)) (= vr2 (fn vr0 vr1 vr2)))))
```

So far, the student has written this code:

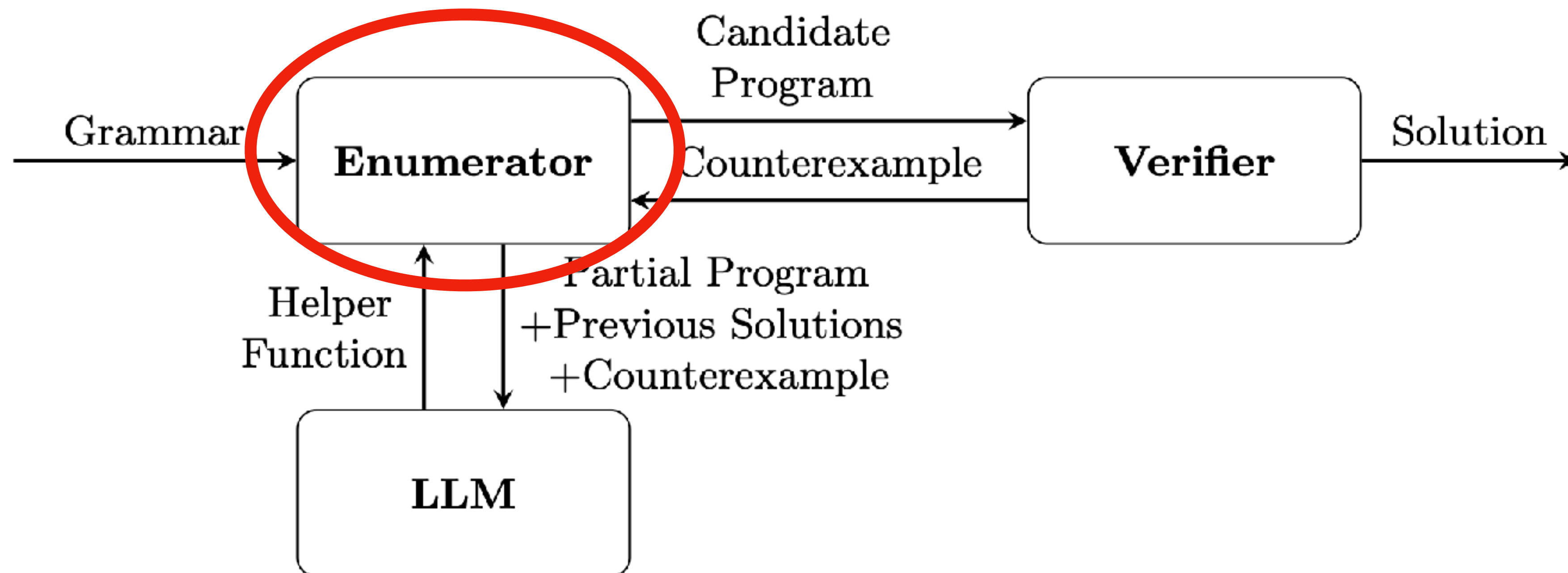
```
(define-fun fn ((vr0 Int) (vr1 Int) (vr2 Int)) Int  
  (ite ?? ?? ??))
```

Can you suggest some helper functions for the student to use to complete this code and replace the ??

1. Initialize weight of each rule.
2. Use the LLM-generated helper functions to augment the grammar appropriately.
3. The response parser also updates the weights of all existing rules in the grammar as before.

Weighted A* Search

Based on W. Lee et. al. (PLDI 2018)



Weighted A* Search

Based on W. Lee et. al. (PLDI 2018)

1. Every time we use the LLM, there is a cost of API call + updating the grammar and $g(x)$ for A*
2. Call LLM after every n steps in the A* search.

A*-iLLM-synth

Results

| Method | BV (384) | LIA (87) | INV (138) | Total (609) |
|--------------------------|--------------|-------------|--------------|----------------------|
| CVC5 | 292.0 | 43.0 | 80.0 | 415.0 (68.1%) |
| LLM Only | 137.0 | 54.0 | 112.0 | 303.0 (49.8%) |
| LLM \cup e-pCFG-synth | 255.0 | 64.0 | 117.7 | 436.7 (71.7%) |
| LLM \cup A*-pCFG-synth | 305.0 | 65.0 | 118.0 | 488.0 (80.1%) |
| A*-iLLM-synth | 272.3 | 68.3 | 67.3 | 408.0 (67.0%) |

average of 3 runs; timeout 600s

Evaluation

1. A*-iLLM-synth is comparable to CVC5 without custom optimizations
2. LLM \cup A*-pCFG-synth is better perhaps because of prompting techniques or the speed of heuristic search over language models
3. Failure Modes: Misunderstandings due to complex constraints, simple errors (applying non-commutative operators to operands in the wrong order, concatenating bit-vectors in the wrong order), hallucinating operations.
4. Benchmarks uniquely solved by the LLM is 4.7x the length of a solution for benchmarks uniquely solved by CVC5. Solutions found by A* contain fewer than 3 operators, but A*-iLLM-synth finds solutions with greater than 20 operators.

Limitations

1. **LLM Training Data:** The SyGuS problems are publicly available.
2. **Reproducibility:** LLMs behave non-deterministically in a way that cannot be seeded. Also dependent on the LLM.
3. **Hyperparameters:** The performance is very sensitive to parameter tuning.
4. **PBE:** LLMs cannot provide guidance to the enumerator for Programming-by-Examples. It tends to provide a solution in the form of a large case split over the input examples.