Expecto:

Extracting Formal Specifications from Natural Language Description for Trustworthy Oracles

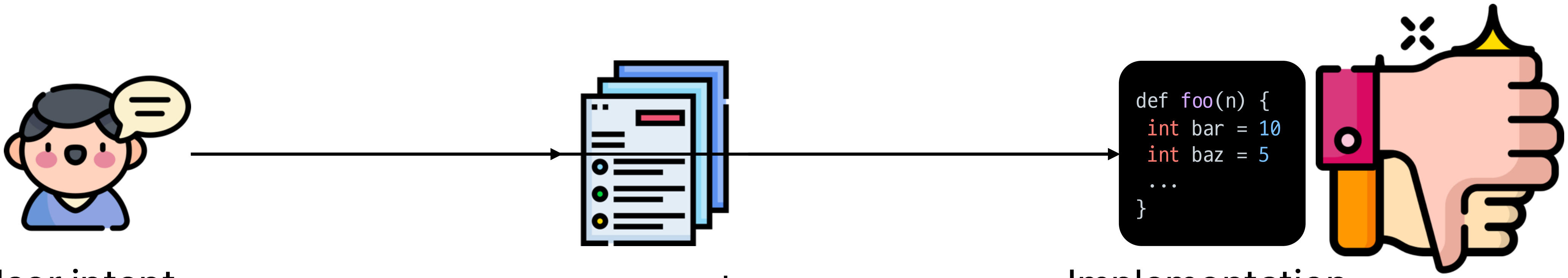
Dongjae Lee (KAIST), Kihong Heo (KAIST)

PLDI BOULDER 2026

Programming Systems Laboratory

KAIST

Spec-Driven Development is Emerging

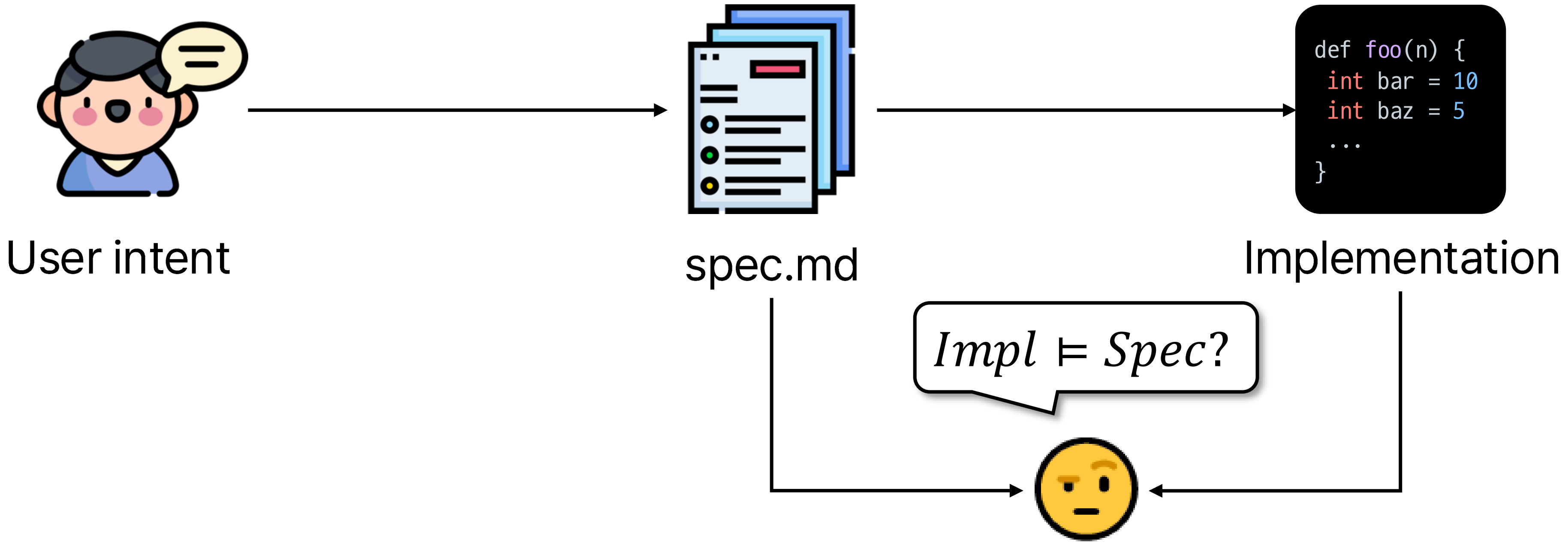


User intent spec.md Implementation

Many of AI coding tools support spec-driven development

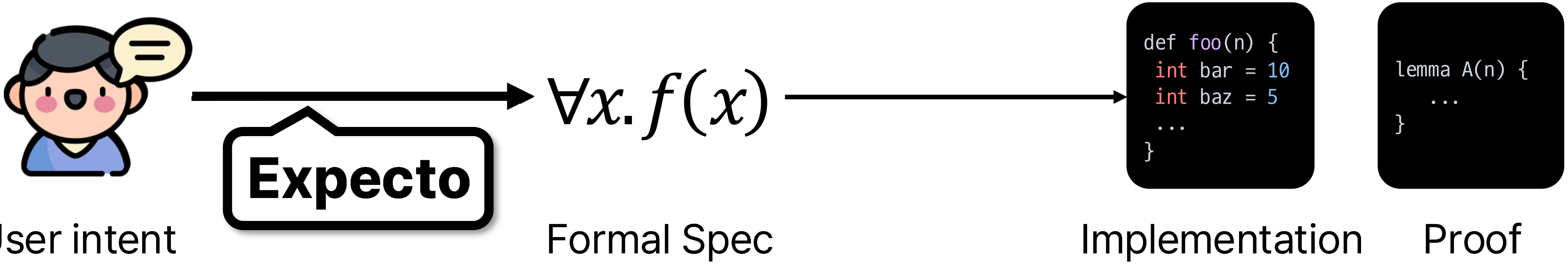
Kiro (Amazon) SpecKit (GitHub) Codex (OpenAI) Claude Code (Anthropic)

Limitation of Spec-Driven Development



- How to check implementation satisfies natural language intent?
- 😭 **Manual Inspection!** **or simply asking an LLM...**

Use Formal Verification! But, ...

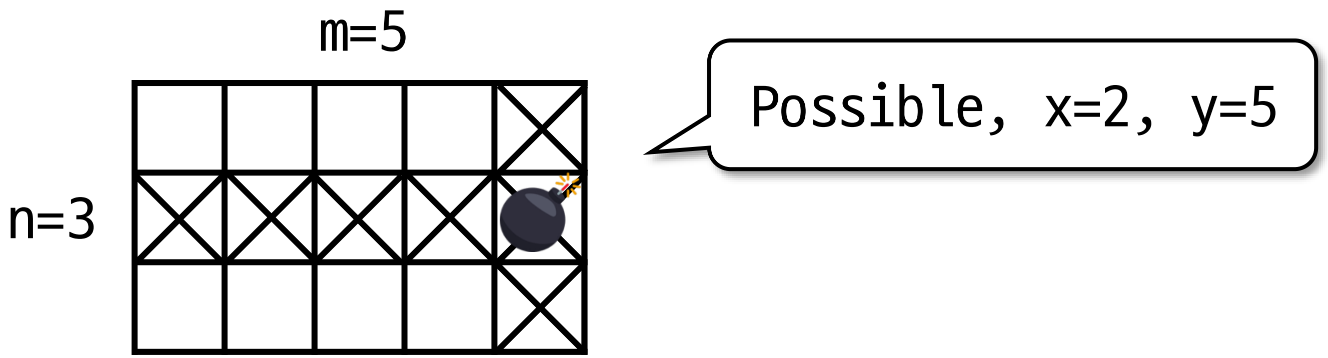


- Problem: writing spec in formal language is challenging
- **Solution: convert natural language intent into formal spec!**

Running Example: A Bomb (Codeforces 699B)

Natural Language Intent

- A **field** is an **n × m** grid consisting of walls (“X”) and empty cells (“ ”).
- If you place a bomb at **(x, y)**, all walls in row **x** and column **y** are removed.
- Determine whether it is possible to remove all walls with a single bomb, and if so, return the values of **x** and **y**.



Problem of Monolithic Spec Generation

- Existing methods generate formal spec at once (Endres et al.)

Wrong condition

```
assert possible == any(
  all((cell[i][x - 1] == "*" or cell[y - 1][j] == "*")
  if cell[i][j] == "*" else True
    for i in range(n)
    for j in range(m))
  for y in range(1, n + 1)
  for x in range(1, m + 1)
) and (
  not possible or
  all((cell[i][x - 1] == "*" or cell[y - 1][j] == "*")
    if cell[i][j] == "*" else True
      for i in range(n)
      for j in range(m))
      for y, x in [(y, x)]
)
```

Duplicated Codes

Unnecessary Loop

Over-simplified condition

```
assert (not possible)
    or (1 <= y <= n and 1 <= x <= m)
```

Expecto Generates Correct & Modular Spec

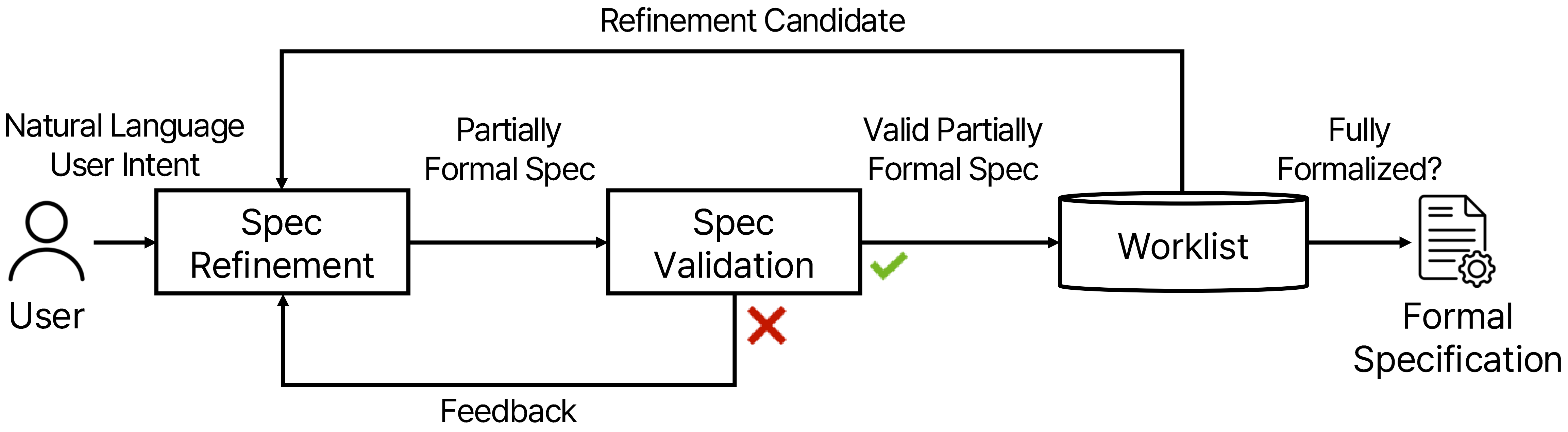
- A field is an n × m grid consisting of walls ("X") and empty cells (" ").
- If you place a bomb at (x, y), all walls in row x and column y are removed.
- Determine whether it is possible to remove all walls with a single bomb, and if so, return the values of x and y.

At most 2.81× more correct specs

Easy to understand specs

- Spec(n, m, field, possible, x, y)= (possible → (ValidPos(x,y,n,m) ∧ WipesAll(x,y,n,m,field))∧ (¬possible → (∀r, c. ValidPos(r,c,n,m) → !WipesAll(r,c,n,m,field))
- ValidPos(r,c,n,m)=(1 ≤ r ≤ n) ∧ (1 ≤ c ≤ m)
- WipesAll(r,c,n,m,field)=∀i,j.IsWall(i, j, n, m, field) → (i+1=r ∨ j+1=c))
- IsWall(r, c, n, m, field)=(0 ≤ r < n) ∧ (0 ≤ c < m) ∧ (field[r][c]="X")

Overview



- **Top-down spec synthesis** generates spec from outline to detail
- **Continuous spec validation** checks partial specs during synthesis

Top-Down Spec Synthesis

- D_i: informal definitions
- D_f: formal definitions

Worklist

D_i `Spec(n, m, field, possible, x, y): "A field is an n × m grid…"`

Spec Refinement (LLM)

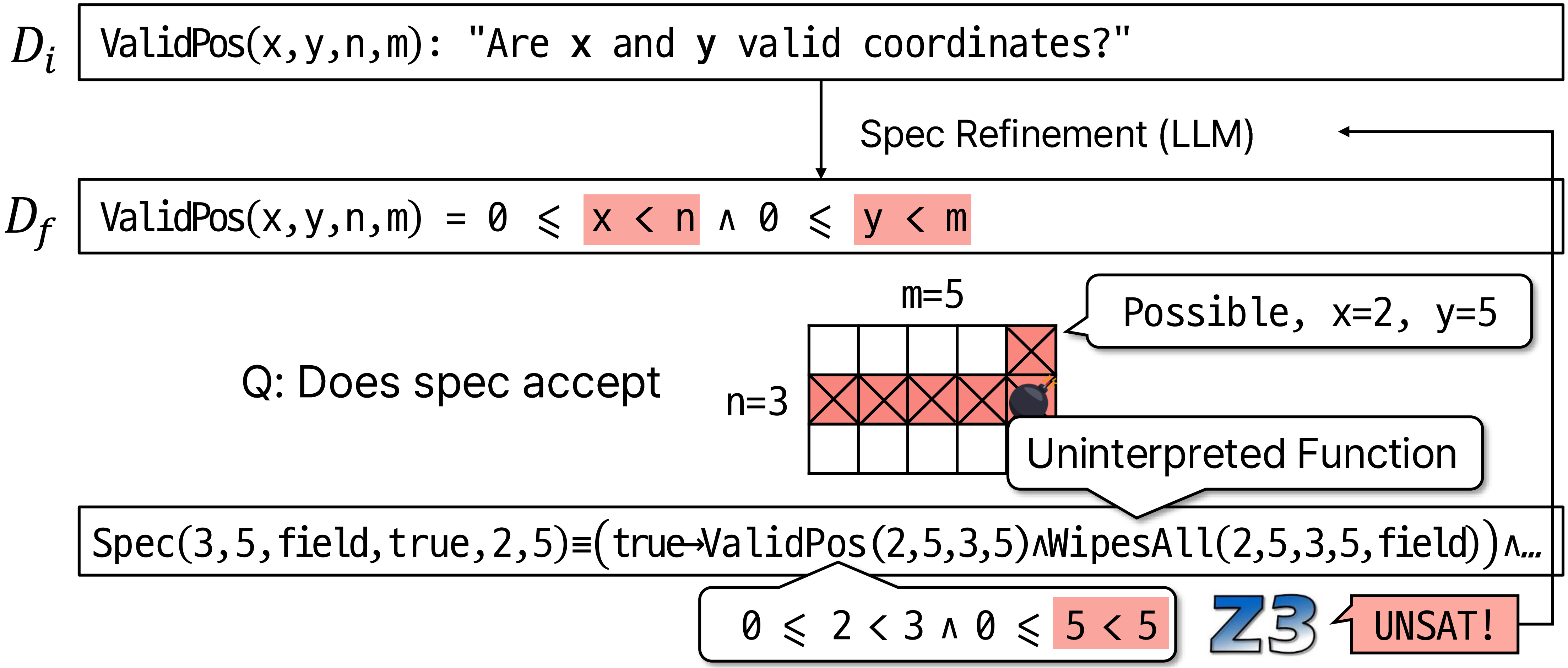
D_f

```
Spec(n, m, field, possible, x, y)=
  (possible → (ValidPos(x,y,n,m) ∧ WipesAll(x,y,n,m,field))∧
  (¬possible → (∀r, c. ValidPos(r,c,n,m) → ¬WipesAll(r,c,n,m,field))
```

D_i

- `ValidPos(x, y, n, m): "Are x and y valid coordinates?"`
- `WipesAll(r, c, n, m, field): "For r and c, will all walls be removed?"`

Continuous Spec Validation



Continuous Spec Validation

D_i `ValidPos(x,y,n,m): "Are x and y valid coordinates?"`

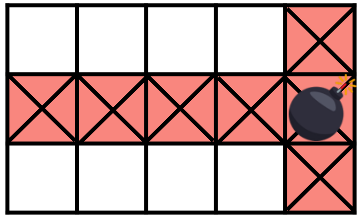
Spec Refinement (LLM)

D_f `ValidPos(x,y,n,m) = 0 < x ⩽ n ∧ 0 < y ⩽ m`

Q: Does spec accept

m=5

n=3



Possible, x=2, y=5

`Spec(3,5,field,true,2,5)≡(true→ValidPos(2,5,3,5)∧WipesAll(2,5,3,5,field))∧…`

`0 < 2 ⩽ 3 ∧ 0 < 5 ⩽ 5`

Z3

SAT!

Types of Validators

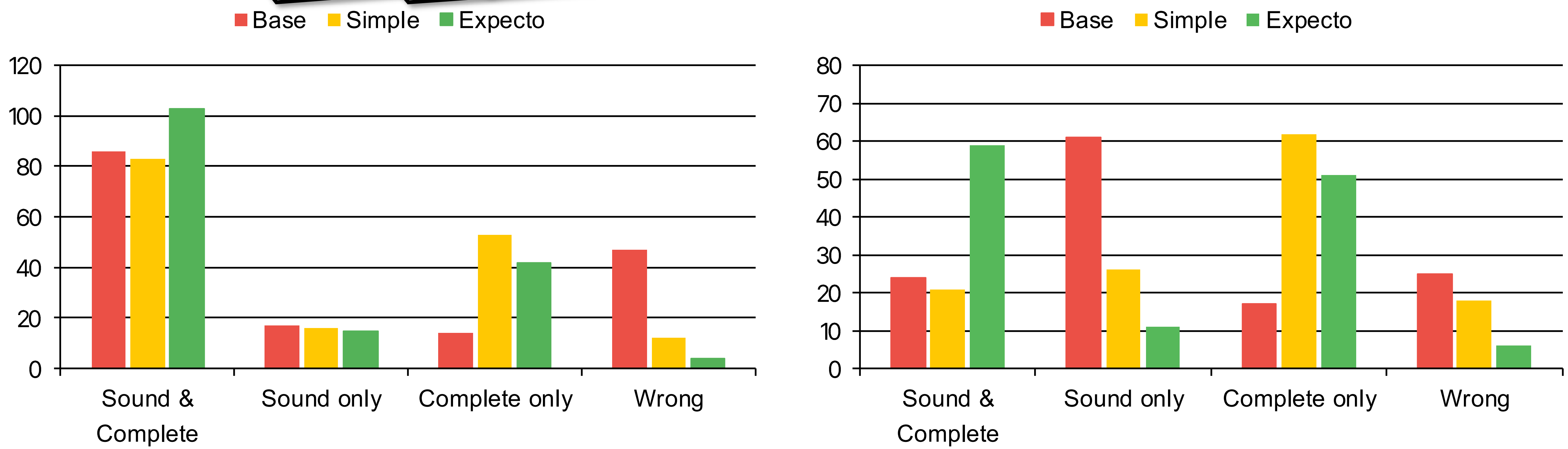
1. Syntax and type validators
2. Logical consistency validator
   - Is spec neither **tautology** nor **contradictory**?
3. Input-output example consistency validator
   - Does spec **accept correct** user-provided input-output examples?
   - Does spec **reject incorrect** user-provided input-output examples?

Experiment Setup

- Benchmark
    - Algorithmic problem: HumanEval+ (Liu et al.), APPS (Hendrycks et al.)
    - Real-world software: Defects4J (Just et al.)
- Baseline
    - NL2Postcond: Monolithic spec generation using LLM (Endres et al.)
- Model
    - GPT-4.1-mini (temperature=0.2)
- SMT Solver
    - Z3

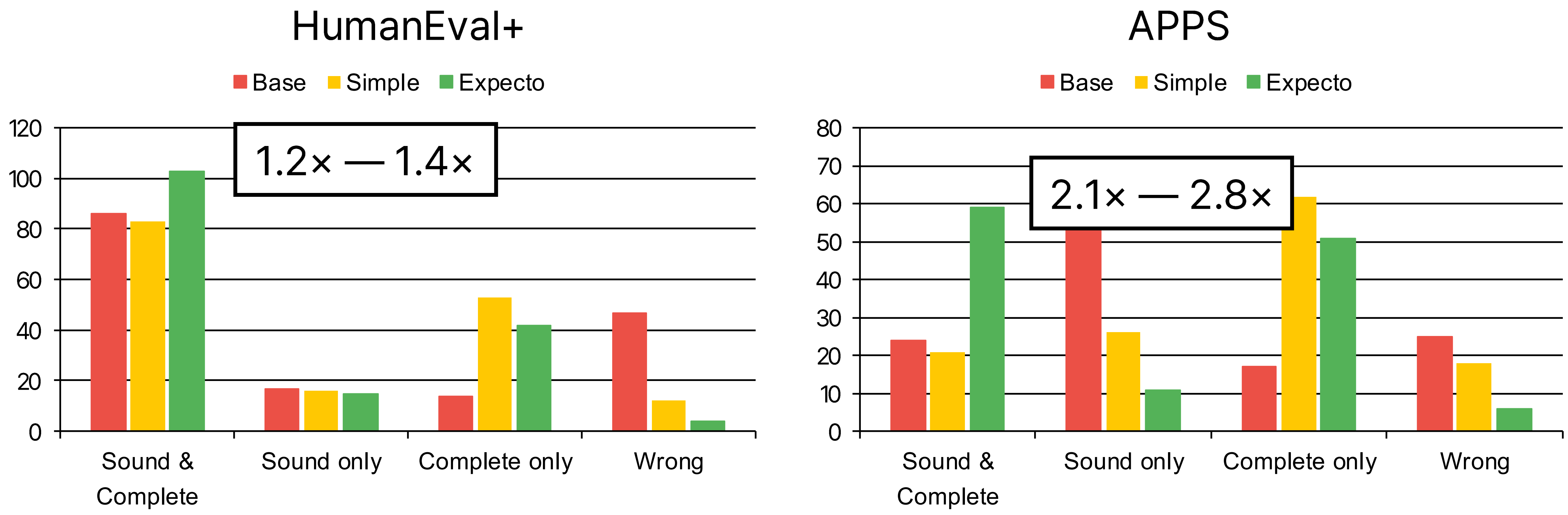
Can Expecto Generate Correct Specs?

NL2P NL2Post Full Expecto capturing a single constraint



Soundness: Rejects incorrect input-output pairs
Completeness: Accepts correct input-output pairs

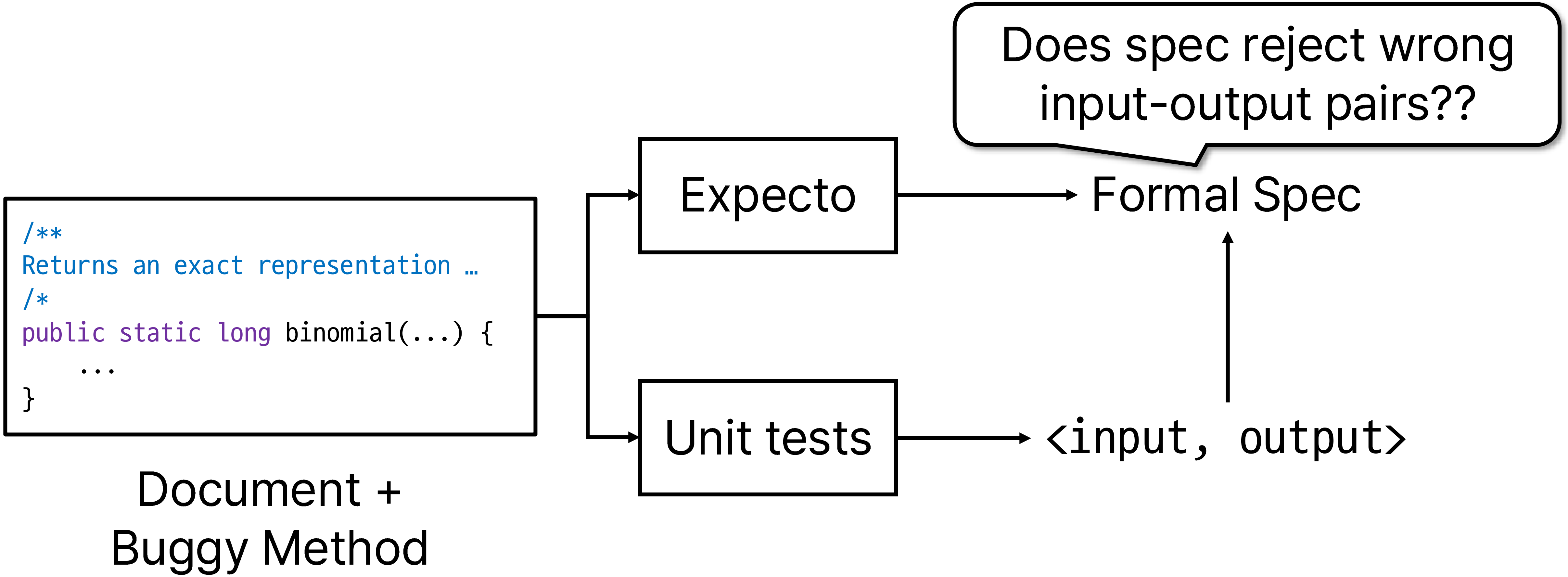
Can Expecto Generate Correct Specs?



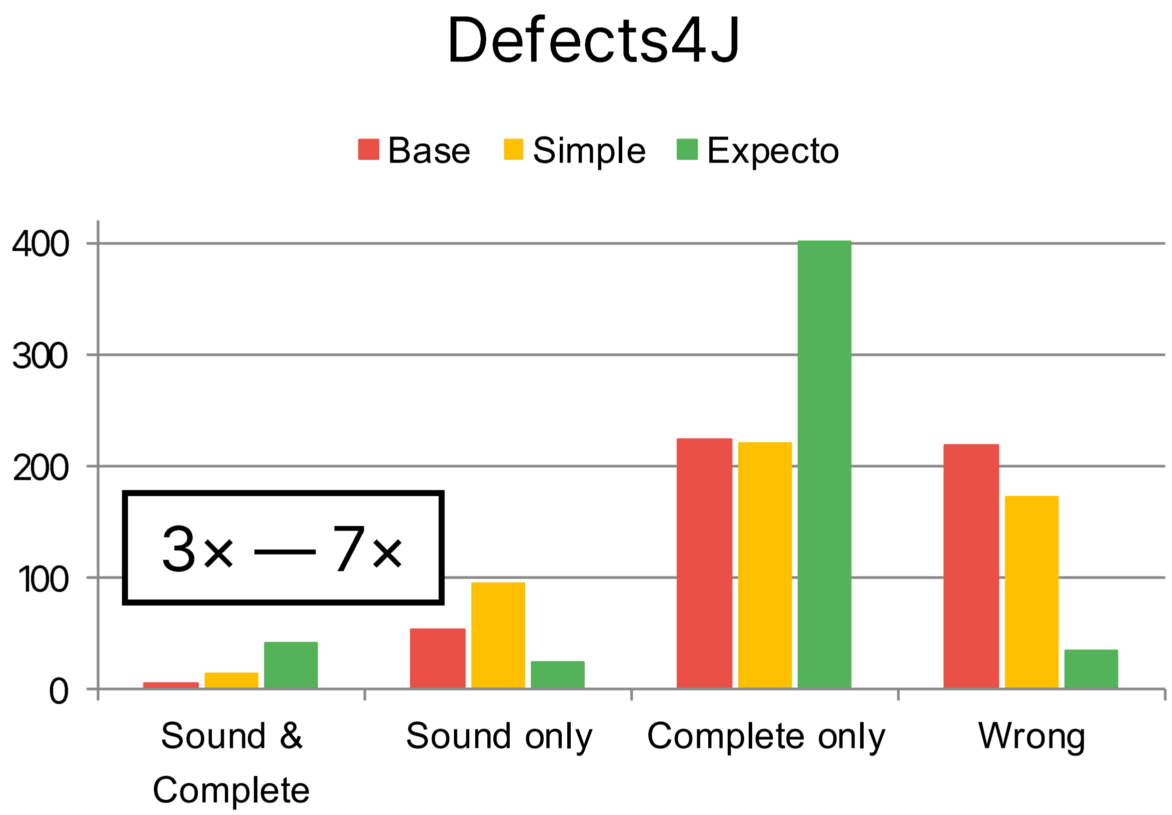
Expecto generates the most correct specs!

Can Expecto Detect Real-World Bugs?

- Detect bugs in Java methods using formal spec from document



Can Expecto Detect Real-World Bugs?



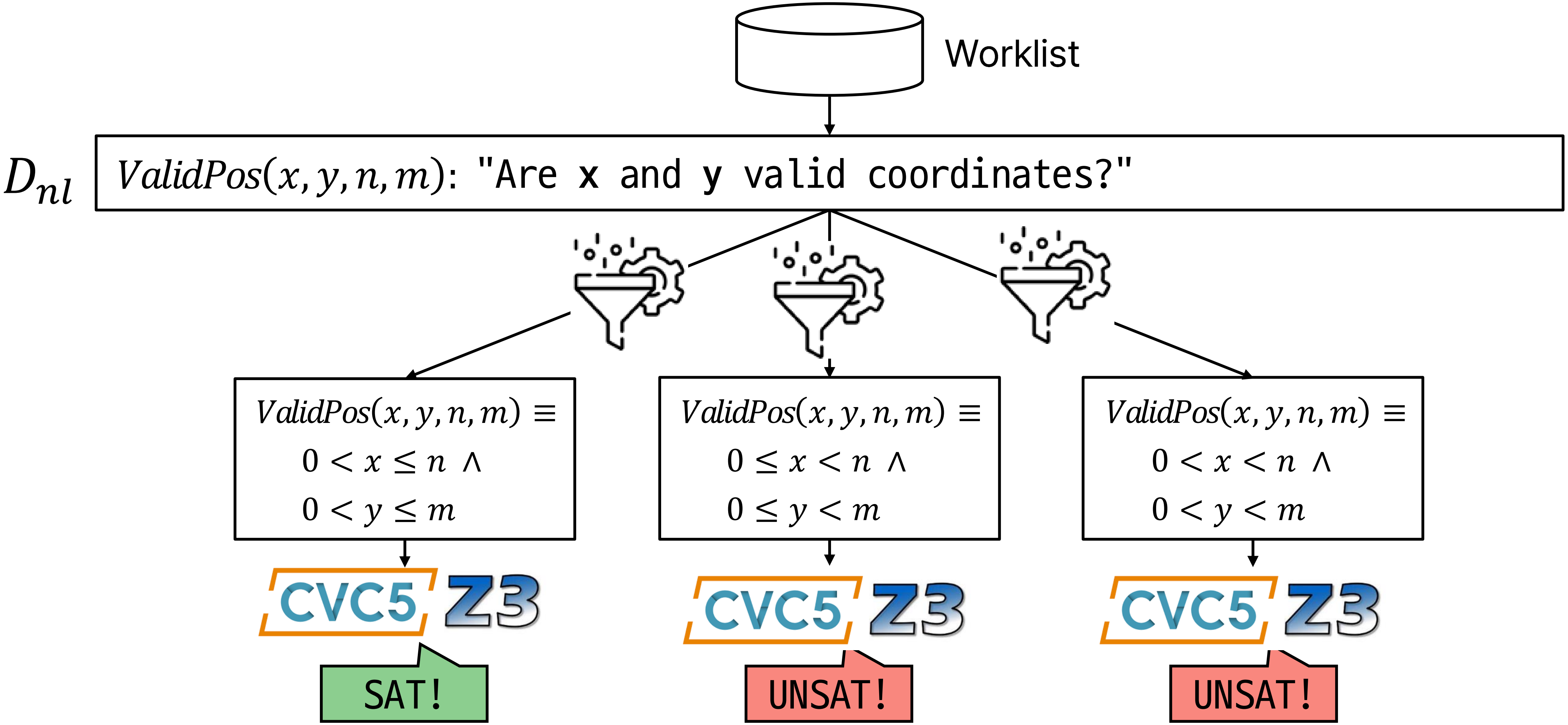
Expecto can detect bugs using generated specs

Summary

- **Problem: formal verification is promising but needs formal spec!**
  - Writing formal spec is challenging
  - Existing monolithic methods are limited
- **Solution: Expecto**
  - Top-down spec synthesis
  - Continuous spec validation
- **Results**
  - Generates more sound & complete specs on algorithmic (at most 2.8×) and real-world software benchmarks (at most 7×)

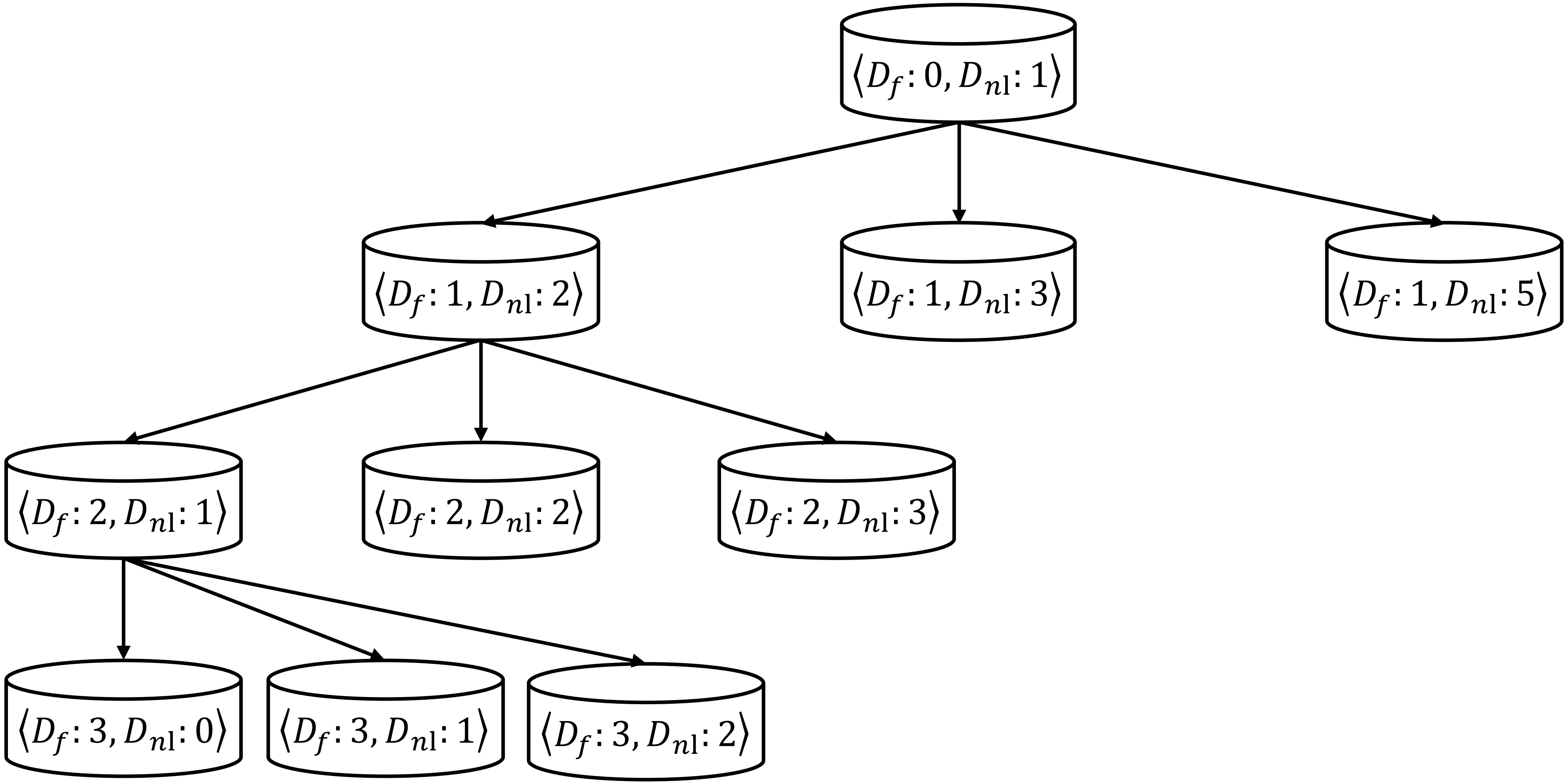
Homepage

Tree Search Explores Diverse Paths



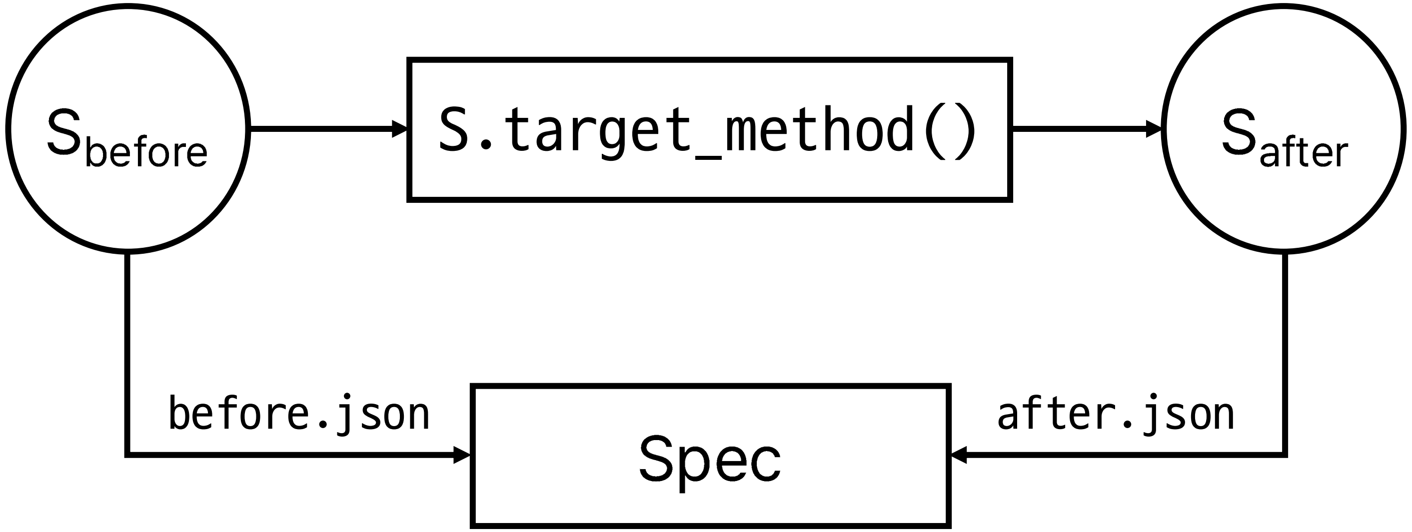
Tree Search prioritization strategy

- Larger D_f, smaller D_{nl} is better!



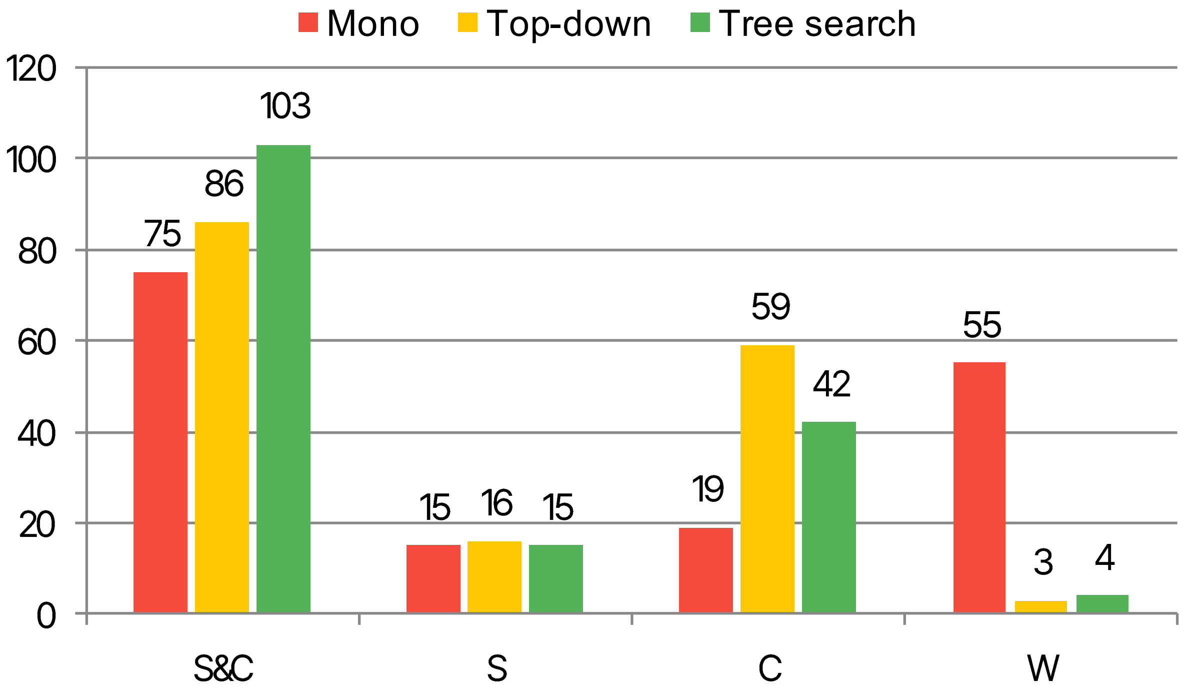
Modeling Side Effects

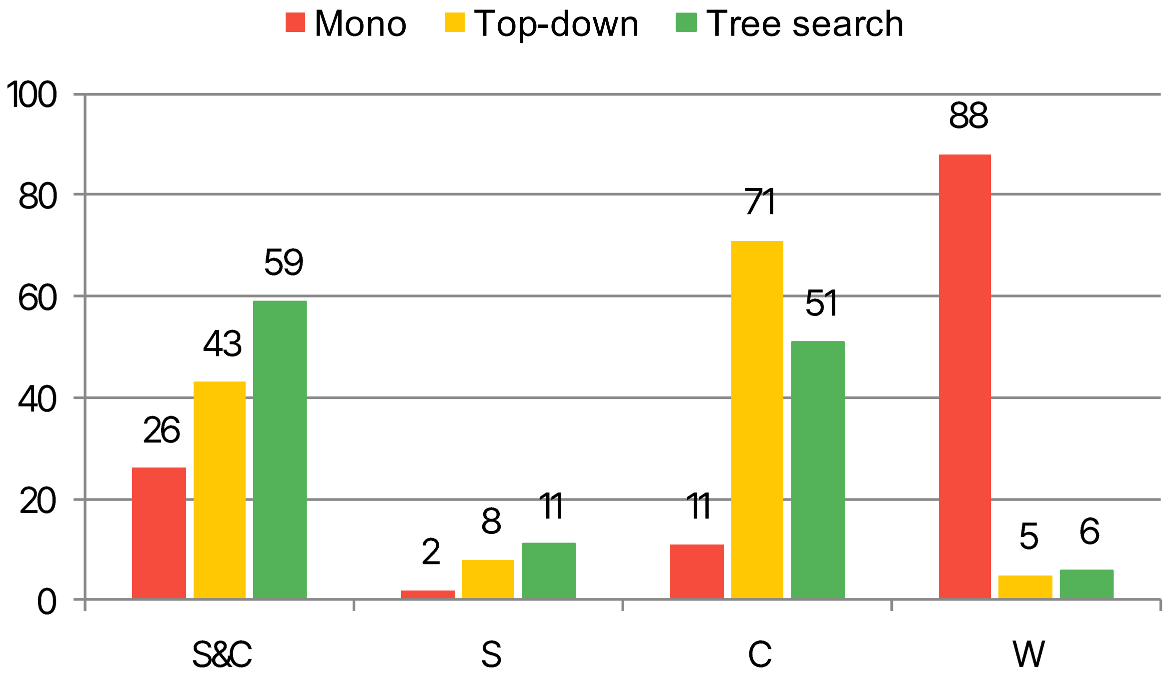
- Detect bugs in Java methods using formal spec from document
- But Java method is not pure!
- Solution: serialize states before and after execution



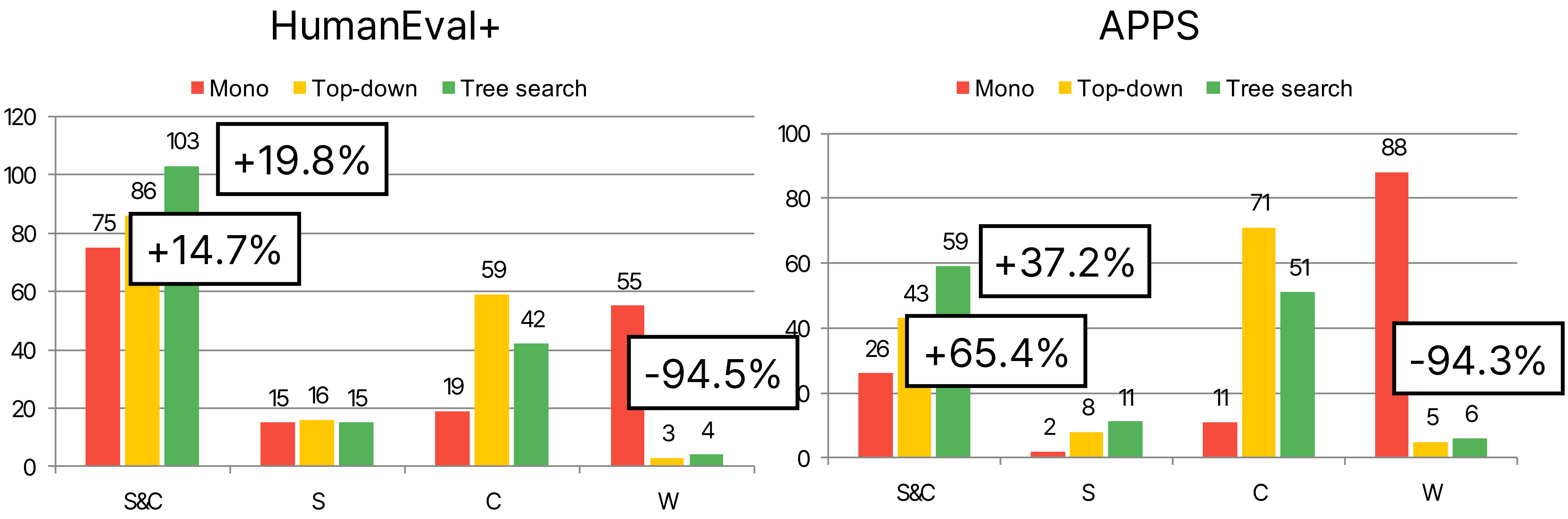
Impact of Search Algorithm

Generate spec with top-down + tree search





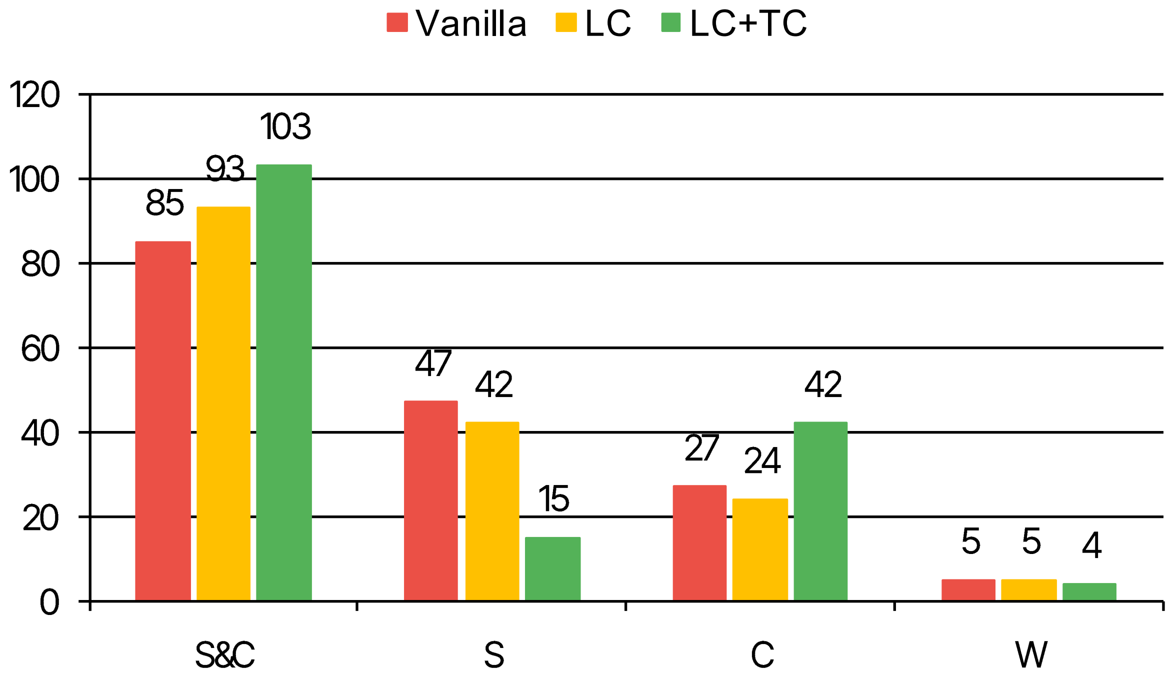
Impact of Search Algorithm



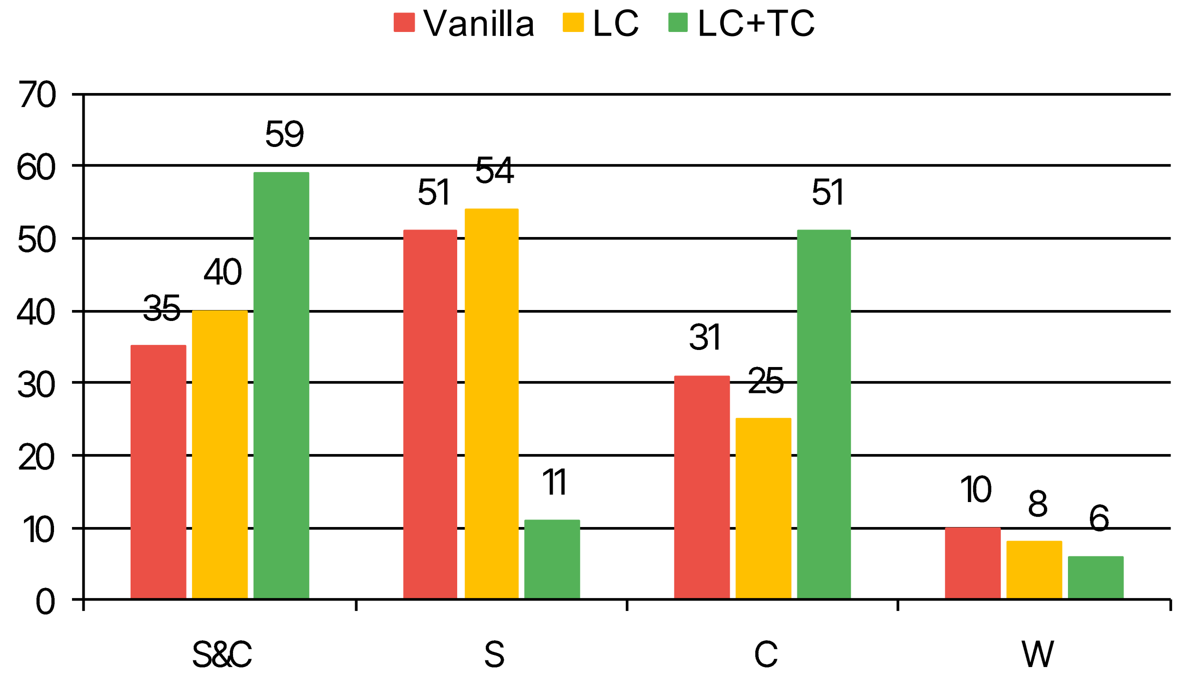
1. **Top-down spec synthesis reduces cognitive burden on LLM**
2. **Tree search explores diverse paths and it leads to more S&C**

Impact of Checkers

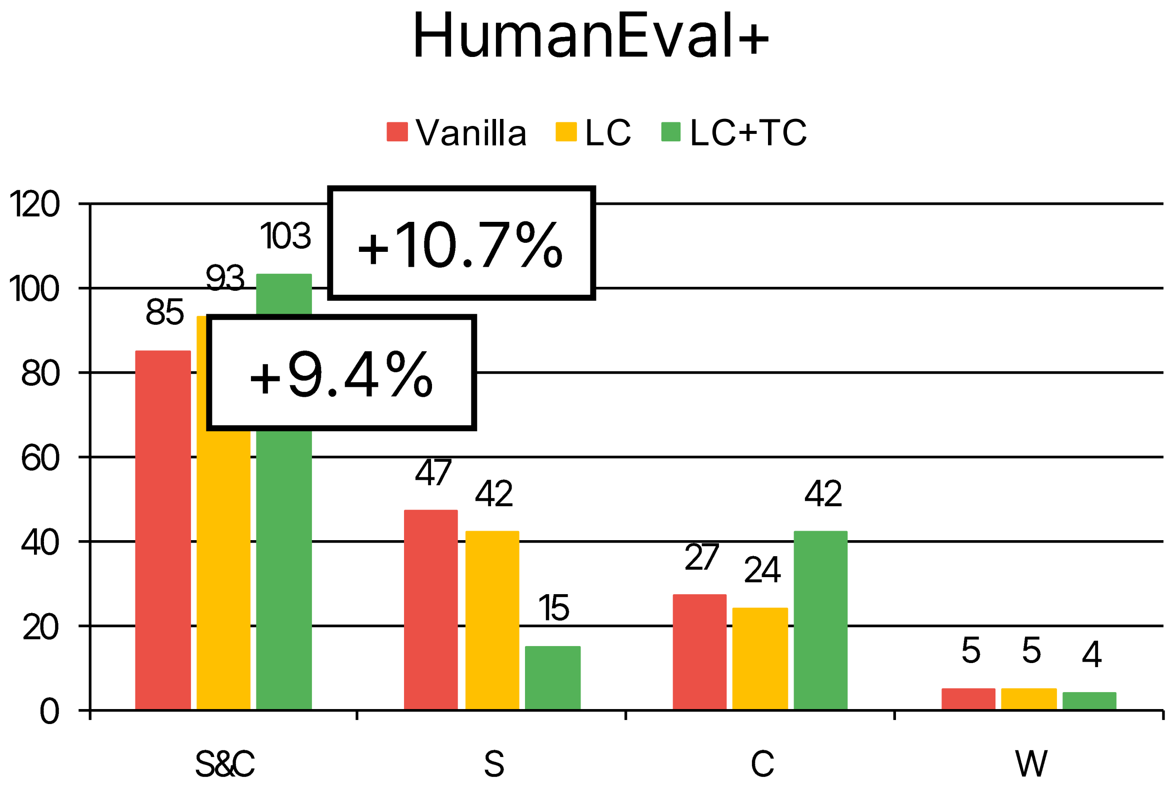
S Van LC + test case consistency checker

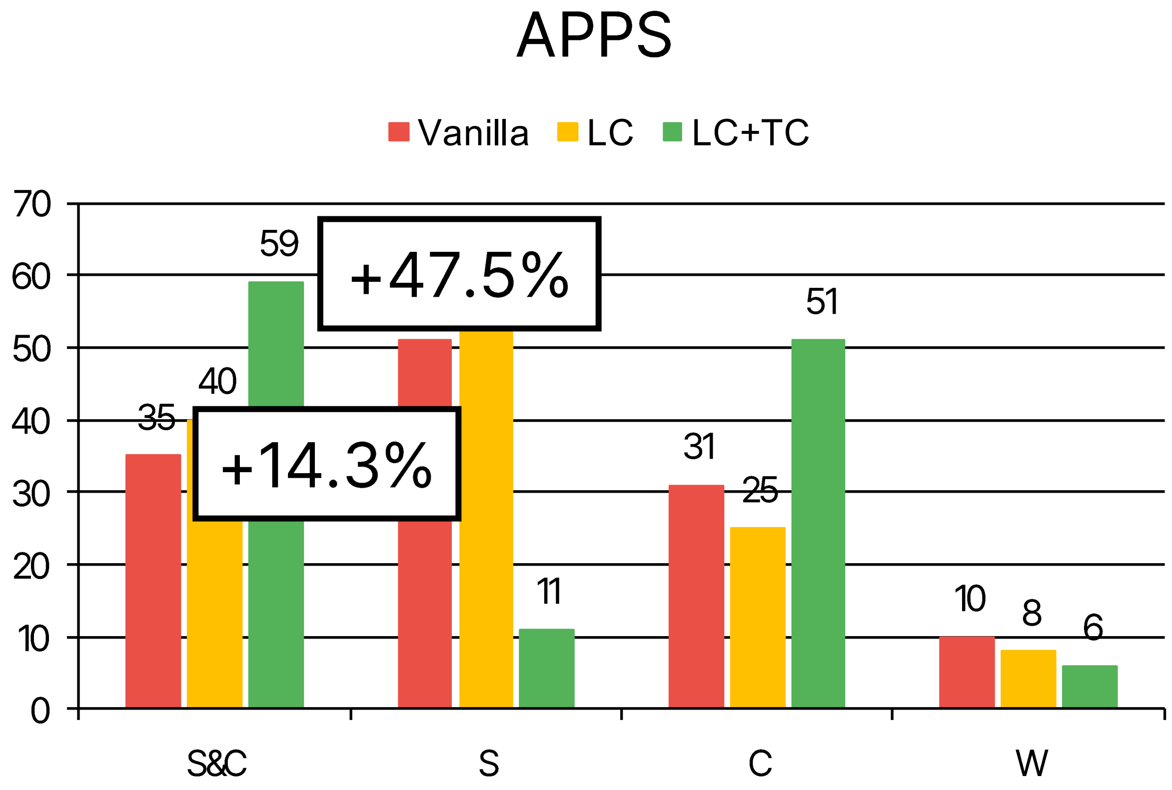


APPS



Impact of Checkers





The More powerful checker derives the more S&C specs

```
def Spec(sq1: list[tuple[int, int]], sq2: list[tuple[int, int]], res: string) {
  var intersect =
    ∃v :: (v in sq1 ∧ InPolygon(sq2, v)) ∨ ∃v :: (v in sq2 ∧ InPolygon(sq1, v)) ∨ EdgesIntersect(sq1, sq2)
  if intersect then (res == "YES") else (res == "NO")
}

def InPolygon(poly: list[tuple[int, int]], pt: tuple[int, int]) {
  var n = len(poly);
  ∃i :: (0 <= i < n ∧ OnSegment(poly[i], pt, poly[((i + 1) % n)])) ∨ (RayIntersections(poly, pt) % 2 == 1)
}

def OnSegment(p: tuple[int, int], q: tuple[int, int], r: tuple[int, int]) {
  var px, py = p; var qx, qy = q; var rx, ry = r;
  (qx - px) * (ry - py) == (qy - py) * (rx - px) ∧
    qx >= (if px < rx then px else rx) ∧ qx <= (if px > rx then px else rx) ∧
    qy >= (if py < ry then py else ry) ∧ qy <= (if py > ry then py else ry)
}

def RayIntersections(poly: list[tuple[int, int]], pt: tuple[int, int]) -> (count: int) {
  var n = len(poly);
  ensure (count == fold(lambda (acc: int, i: int) =
    if RayIntersectsSegment(poly[i], poly[((i + 1) % n)], pt) then (acc + 1) else acc), 0, [0 .. (n - 1)]);
}

def RayIntersectsSegment(p1: tuple[int, int], p2: tuple[int, int], q: tuple[int, int]) {
  var px, py = q; var x1, y1 = p1; var x2, y2 = p2;
  var y_min = (if y1 < y2 then y1 else y2); var y_max = (if y1 > y2 then y1 else y2);
  var x_min = (if x1 < x2 then x1 else x2); var x_max = (if x1 > x2 then x1 else x2);
  var segment_horizontal = (y1 == y2);
  var point_on_horizontal_segment = (segment_horizontal ∧ (py == y1)) ∧ (px >= x_min) ∧ (px <= x_max);
  var intersects_right =
    ((¬ segment_horizontal) ∧ (py >= y_min) ∧ (py <= y_max) ∧
    ((x1 + (((py - y1) * (x2 - x1)) / (y2 - y1))) >= px));
  (point_on_horizontal_segment ∨ intersects_right)
}

def EdgesIntersect(p1: list[tuple[int, int]], p2: list[tuple[int, int]]) {
  var n1 = len(p1); var n2 = len(p2);
  ∃(i: int, j: int) :: ((0 <= i < n1) ∧ (0 <= j < n2) ∧ EdgePairIntersect(p1[i], p1[((i + 1) % n1)], p2[j], p2
      [((j + 1) % n2)]))
}

def EdgePairIntersect(p1: tuple[int, int], p2: tuple[int, int], q1: tuple[int, int], q2: tuple[int, int]) {
  var o1 = Orientation(p1, p2, q1); var o2 = Orientation(p1, p2, q2);
  var o3 = Orientation(q1, q2, p1); var o4 = Orientation(q1, q2, p2);
  (o1 != o2 ∧ o3 != o4) ∨
    (o1 == 0 ∧ OnSegment(p1, q1, p2)) ∨ (o2 == 0 ∧ OnSegment(p1, q2, p2)) ∨
    (o3 == 0 ∧ OnSegment(q1, p1, q2)) ∨ (o4 == 0 ∧ OnSegment(q1, p2, q2))
}

def Orientation(p: tuple[int, int], q: tuple[int, int], r: tuple[int, int]) -> (o: int) {
  var px, py = p; var qx, qy = q; var rx, ry = r;
  var val = (qy - py) * (rx - qx) - (qx - px) * (ry - qy);
  ensure (val == 0 ==> o == 0);
  ensure (val > 0 ==> o == 1);
  ensure (val < 0 ==> o == 2);
}
```

Fig. 13. Full definition of the generated specification for the example problem in Fig. 7