清華大學
Tsinghua University

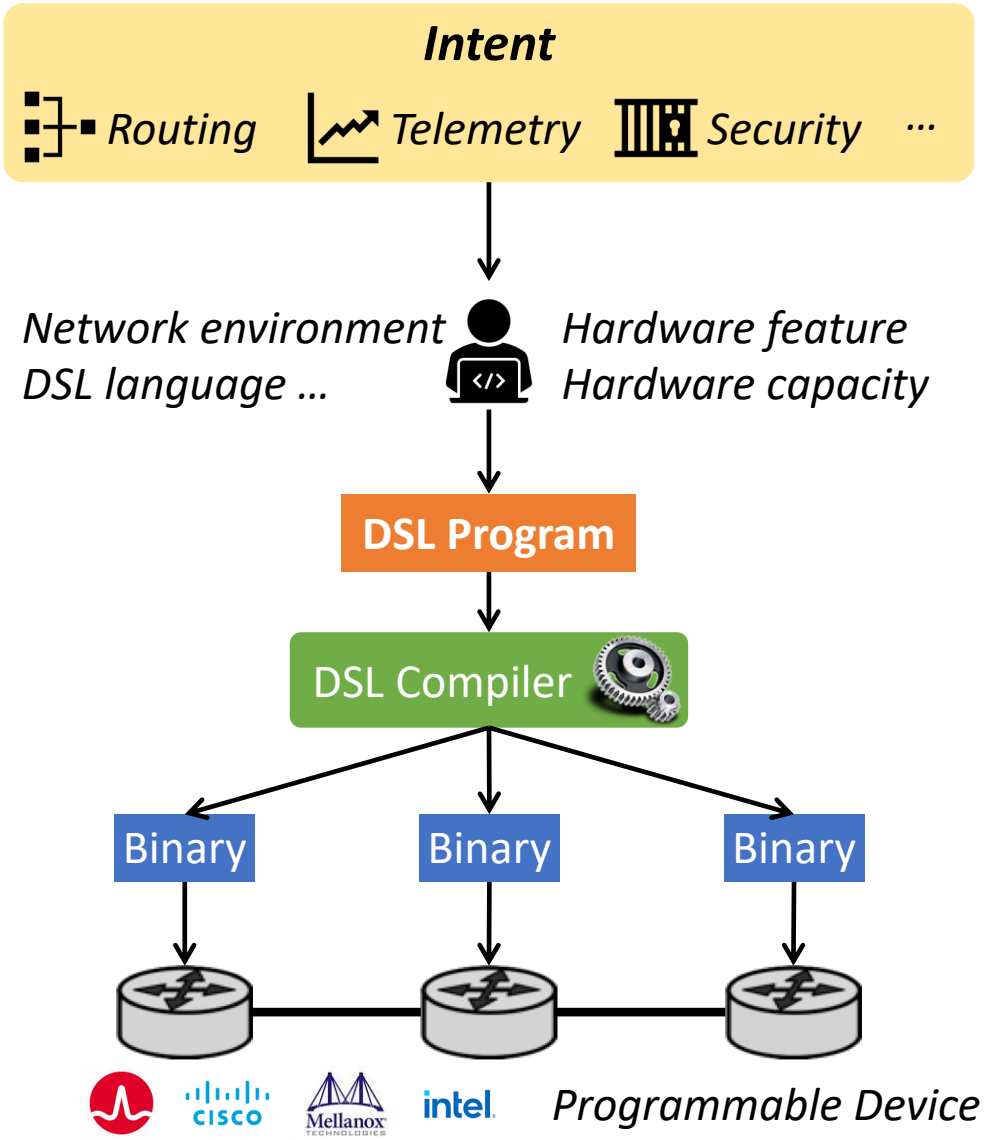
Alibaba Cloud | 
Worldwide Cloud Services Partner

# Finding Bugs in Programmable Data Plane Generators

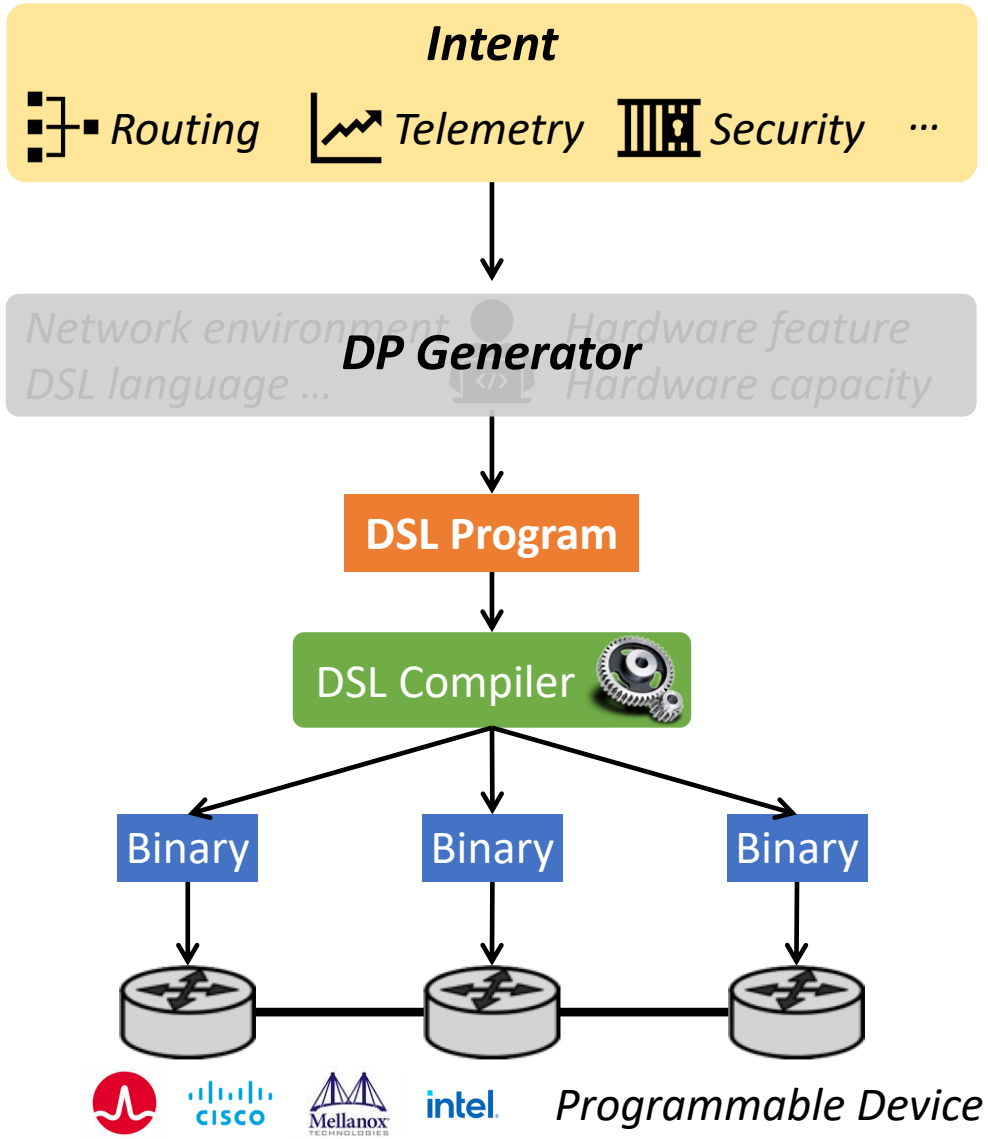
*Jiamin Cao*

Yu Zhou, Chen Sun, Lin He, Zhaowei Xi, Ying Liu

# Programmable Data Plane Gains Significant Traction



# Data Plane Generators Make Programming Easier




**Contra: A Programmable System for Performance-aware Routing**

Kuo-Feng Hsu Ryan Beckett Ang Chen

**Programmable In-Network Security for Context-aware BYOD Policies**

Qiao Kang Lei Xue Adam Morrison

**Poseidon: Mitigating Volumetric DDoS Attacks with Programmable Switches**

**Language-Directed Hardware Design for Network Performance Monitoring**

**Quantitative Network Monitoring with NetQRE**

Yifei Yuan
University of Pennsylvania
vifeiy@cis.upenn.edu

Dong Lin
LinkedIn Inc.
dolin@linkedin.com

Ankit Mishra
University of Pennsylvania
mankit@seas.upenn.edu

**Sonata: Query-Driven Streaming Network Telemetry**

Arpit Gupta
Princeton University

Rob Harrison
Princeton University

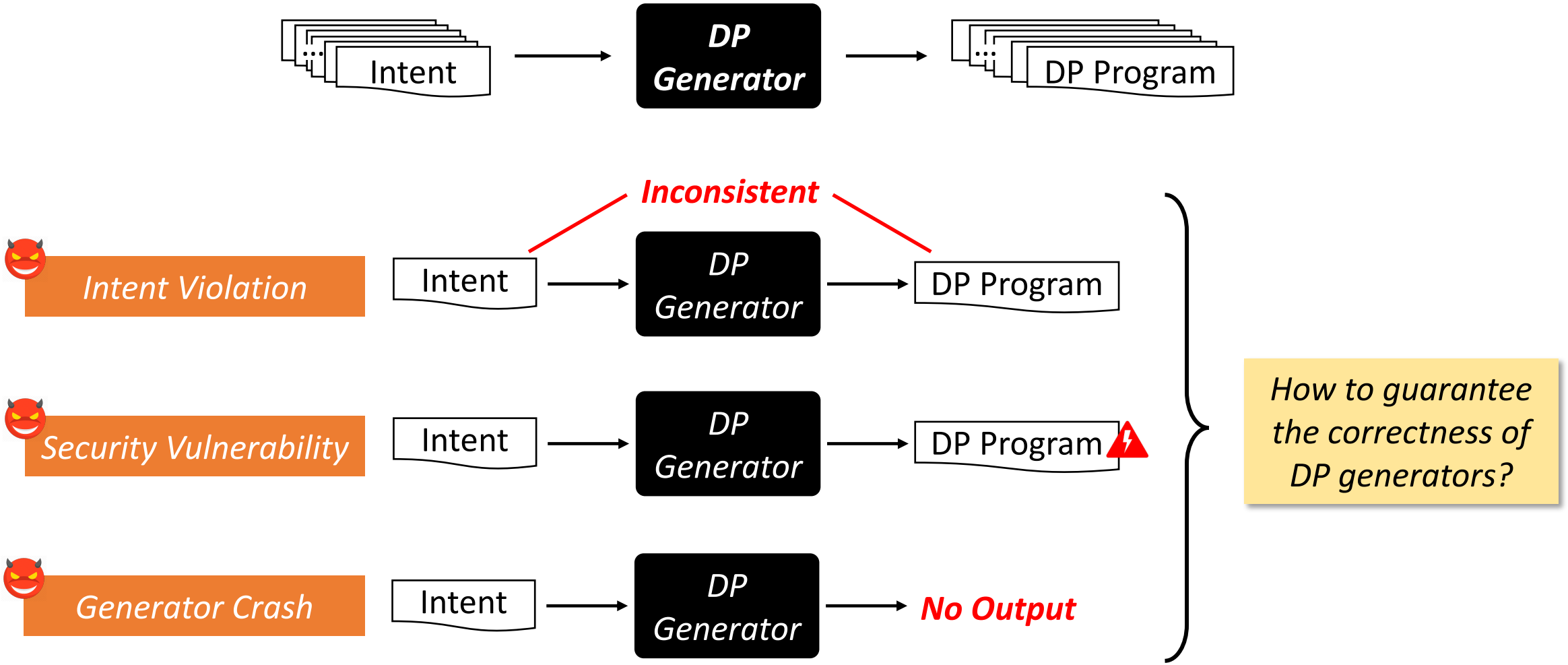
Marco Canini
KAUST

**Lyra: A Cross-Platform Language and Compiler for Data Plane Programming on Heterogeneous ASICs**

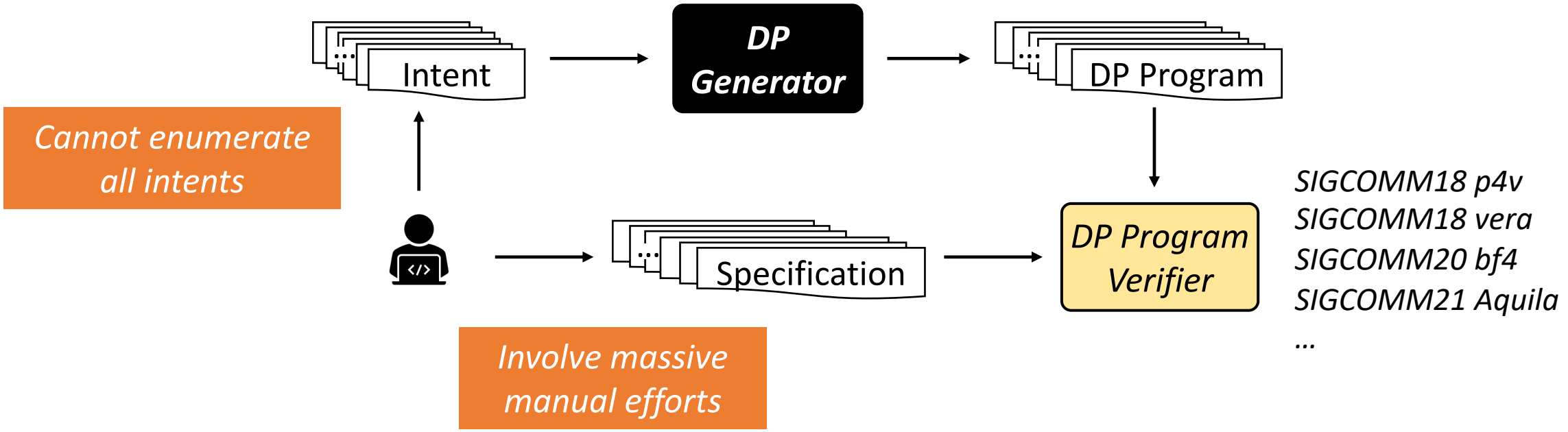
Jiaqi Gao[†§], Ennan Zhai[†], Hongqiang Harry Liu[†], Rui Miao[†], Yu Zhou[†⋄], Bingchuan Tian[†⋆], Chen Sun[†]
Dennis Cai[†], Ming Zhang[†], Minlan Yu[§]

[†]*Alibaba Group* [§]*Harvard University* [⋄]*Tsinghua University* [⋆]*Nanjing University*

# Understanding the Correctness of DP Generators

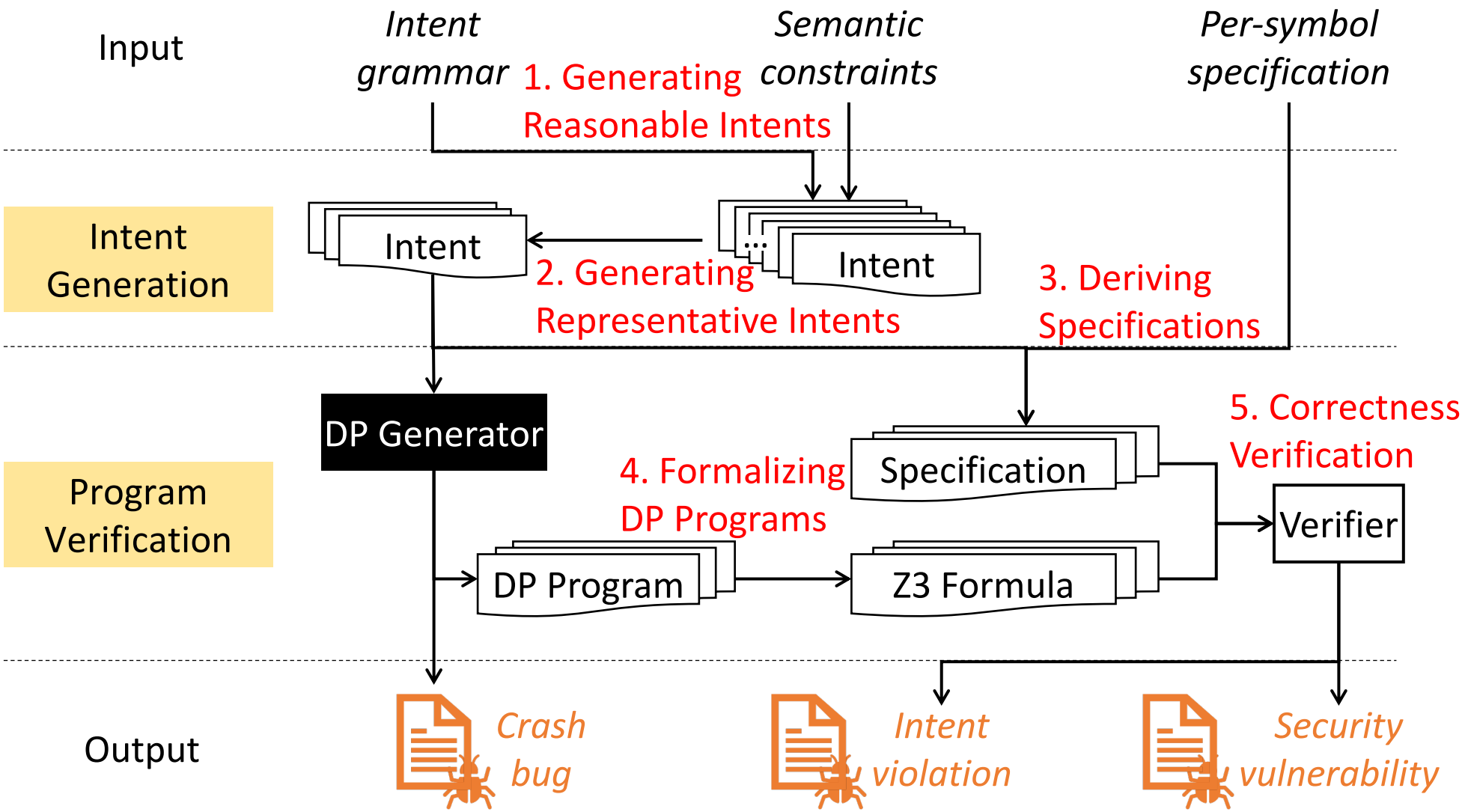


# *Existing Work is Not Designed to Debug DP Generators*

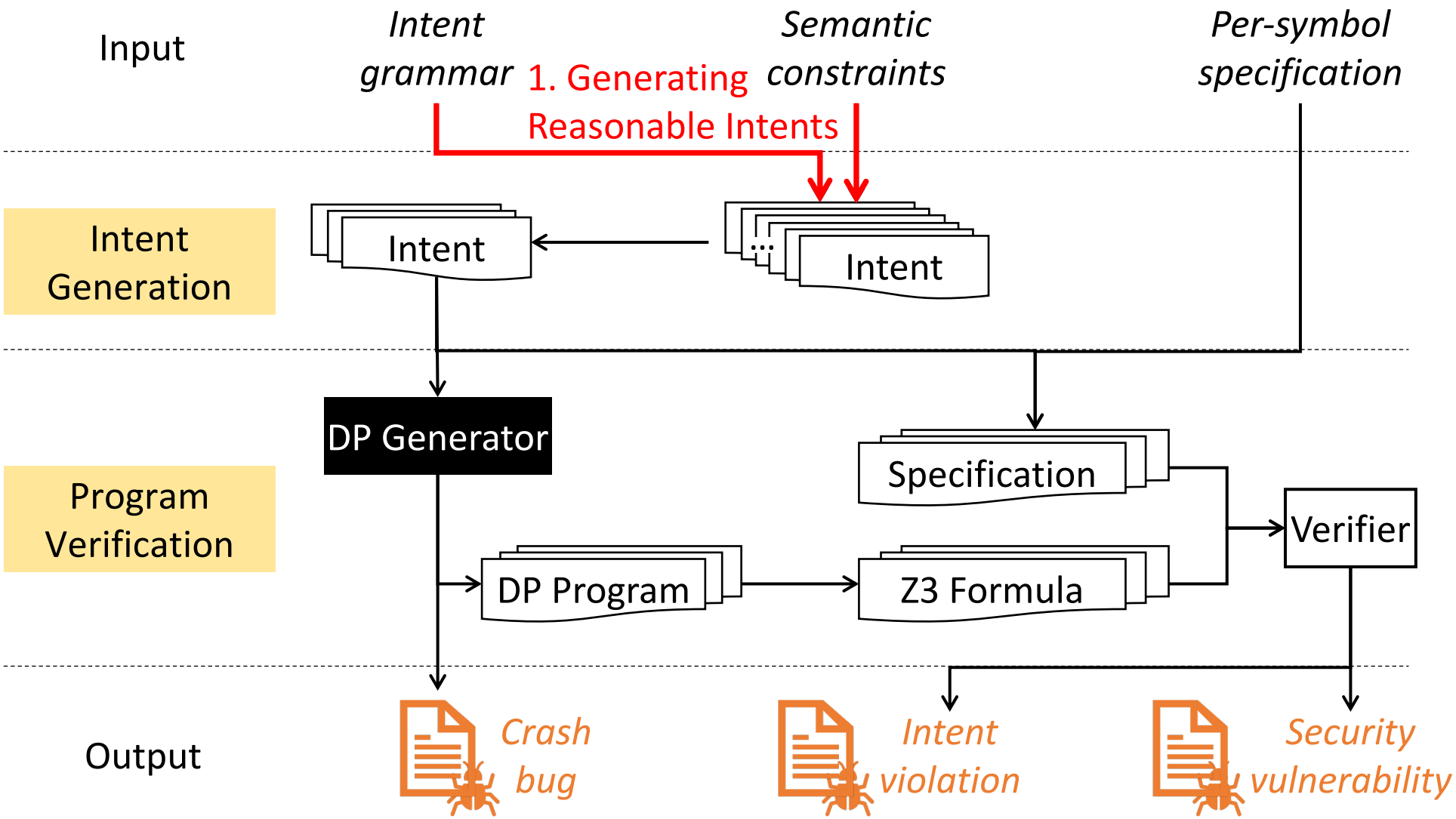


# Firebolt: Finding Bugs in Programmable DP Generators

# Firebolt Workflow



# Firebolt Workflow



# Generating Reasonable Intents

*Syntax-Guided Intent Generation* ⇨ *Syntactically-Correct Intents*

**Intent generation graph**

① ***Start from start symbol***

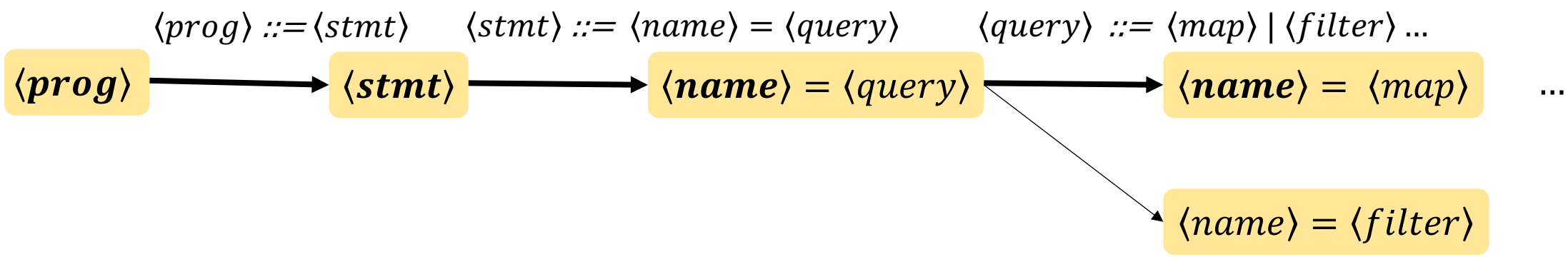
② ***Grow graph using expansion rules***

③ ***Collect leaf nodes as output intents***

Start Symbol  Expansion Rule

$\langle \boldsymbol{prog} \rangle \quad ::= \quad \langle stmt \rangle$
$\langle stmt \rangle \quad ::= \quad \langle name \rangle = \langle query \rangle$
$\langle query \rangle \quad ::= \quad \langle map \rangle \,|\langle filter \rangle$

*BNF Grammar Example*



# Generating Reasonable Intents

*Semantic Constraint Enforcement* ⇨ *Semantically-Valid Intents*

## *Invalid Semantics*

### *Undefined reference*

```
def R2 = func1 (R1, ...);
// R1 is not defined
```

### *Repeated definition*

```
def R2 = func1 (...);
def R2 = func2 (...);
```

## *Context-Aware Semantic Constraint*

*Dependency-type:*

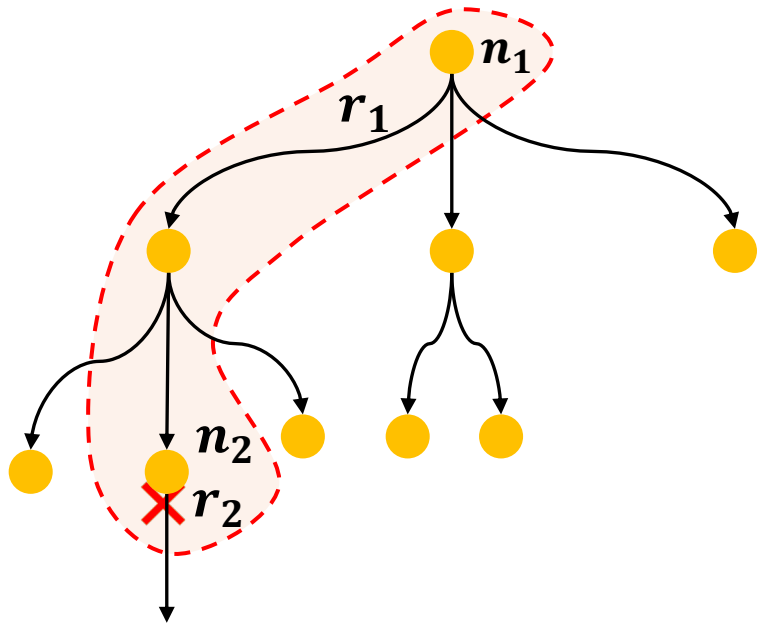
$if\ \exists\ \langle r1\rangle\ on\ \langle n1\rangle, \exists\ \langle r2\rangle\ on\ \langle n2\rangle$

variable declaration — variable definition

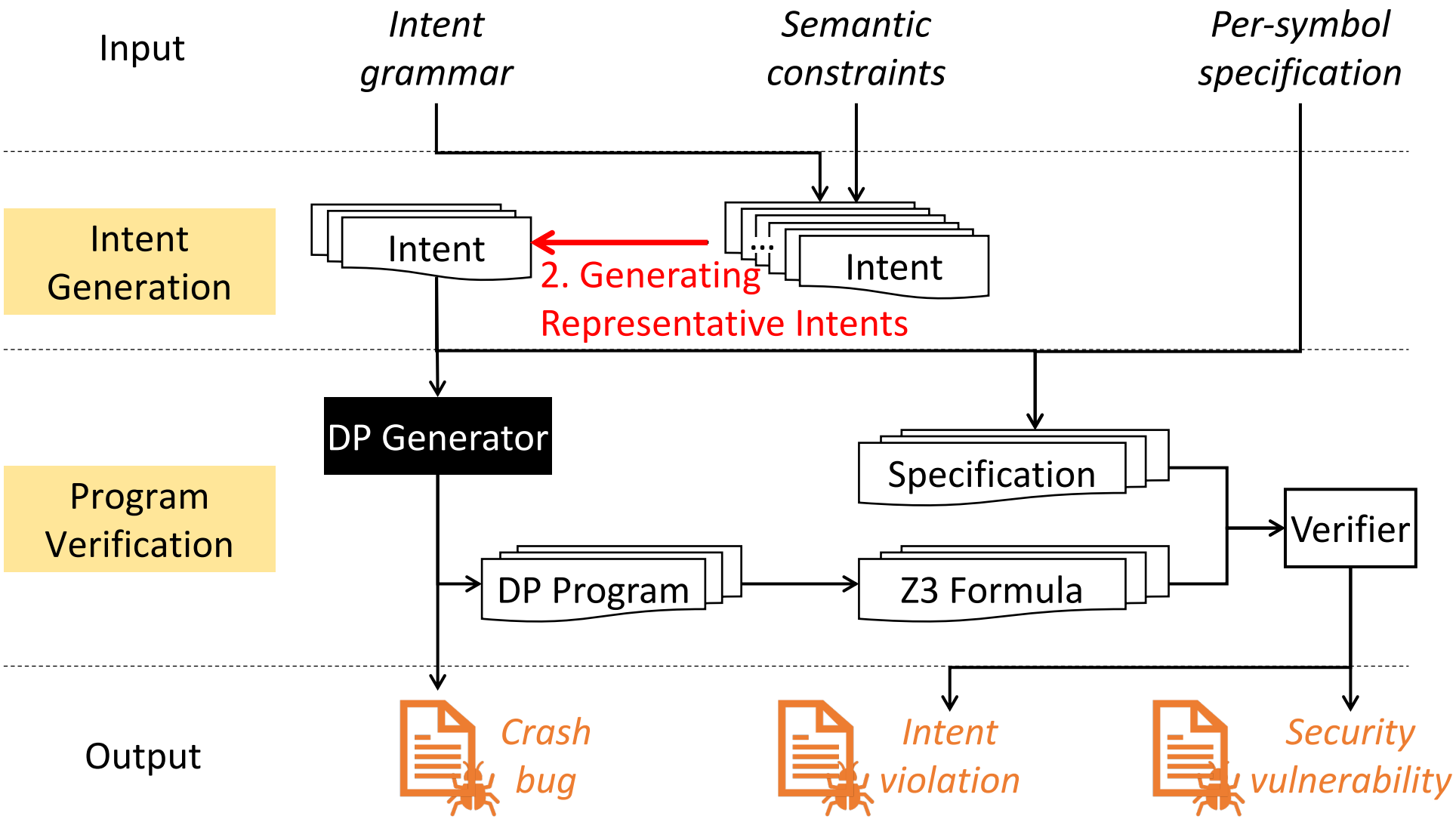
*Exclusion-type:*

$if\ \exists\ \langle r1\rangle\ on\ \langle n1\rangle, \nexists\ \langle r2\rangle\ on\ \langle n2\rangle$

variable definition — (same) variable definition



# Firebolt Workflow

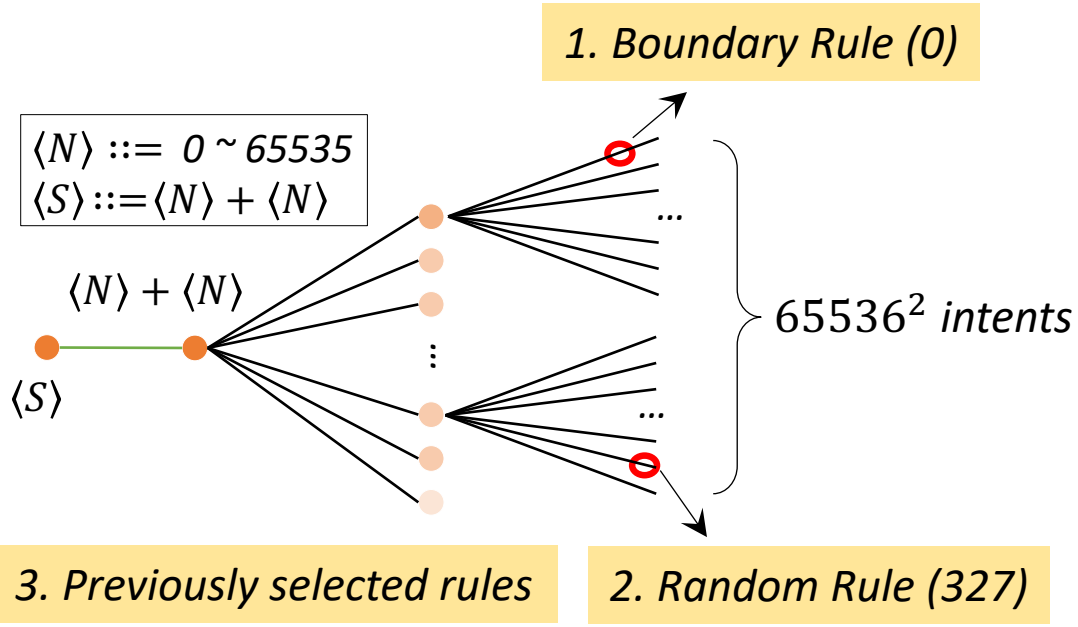


# Generating Representative Intents

*Wide parameter range (many)*

⇩

*Keep representative rules (few)*

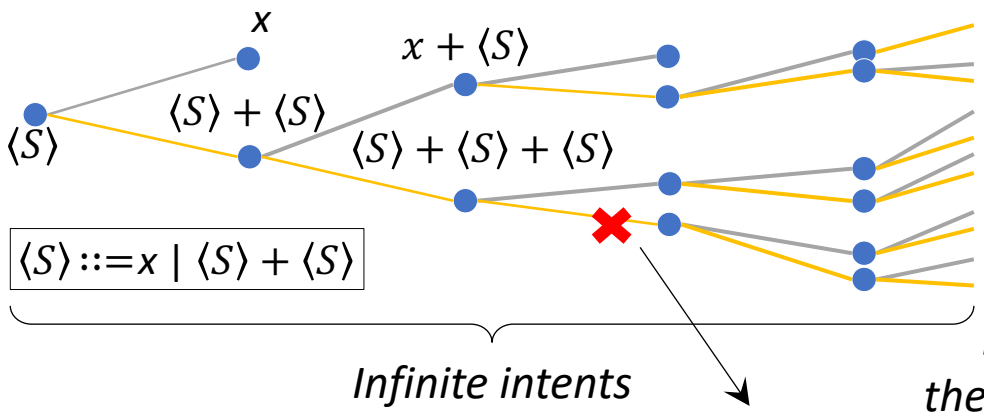



---

*Cyclic symbol reference (infinite)*

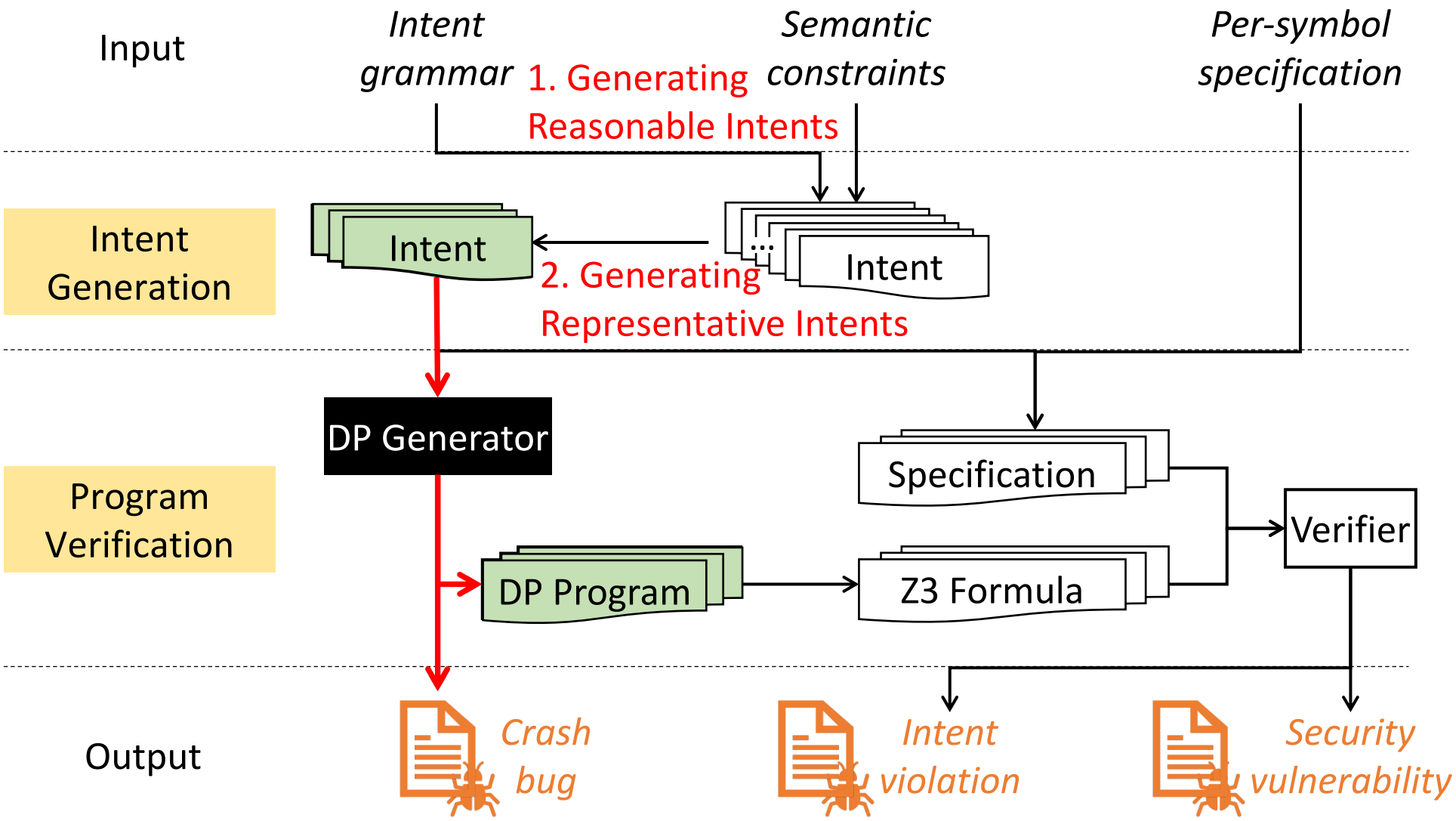
⇩

*Break symbol recurrence (finite)*

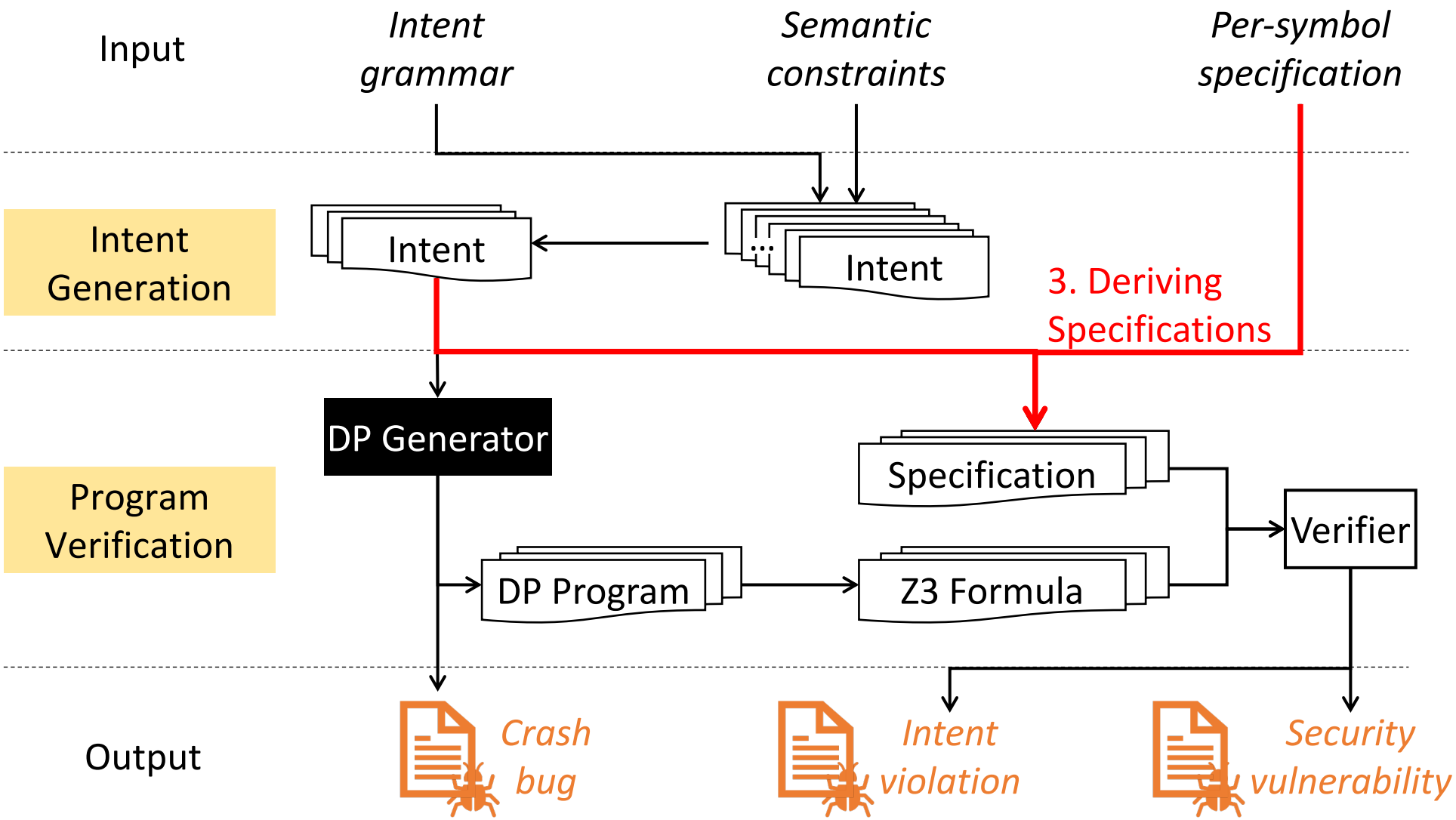



*Based on combinatorial theory, see more in our paper!*

# Firebolt Workflow



# Firebolt Workflow



# Deriving Specifications Automatically

*Observation*

Intents are generated by expanding grammar symbols

*Key idea*

Write specifications of each grammar symbol ⇨ Compose symbol specifications

*General and flexible format:* DEC_FUNC | EXEC_FUNC

*A simple example (counter)*

$\langle count\_stmt \rangle ::= count(\langle expression \rangle)$

```
def DEC_FUNC:
  counter = 0
```

```
def EXEC_FUNC:
  if (⟨expression⟩.exec() == true):
    counter = counter + 1
  return counter
```

$\langle expression \rangle ::= \langle field \rangle == \langle value \rangle$

```
def DEC_FUNC:
  counter = 0
```

```
def EXEC_FUNC:
  return ⟨field⟩.exec() == ⟨value⟩.exec()
```

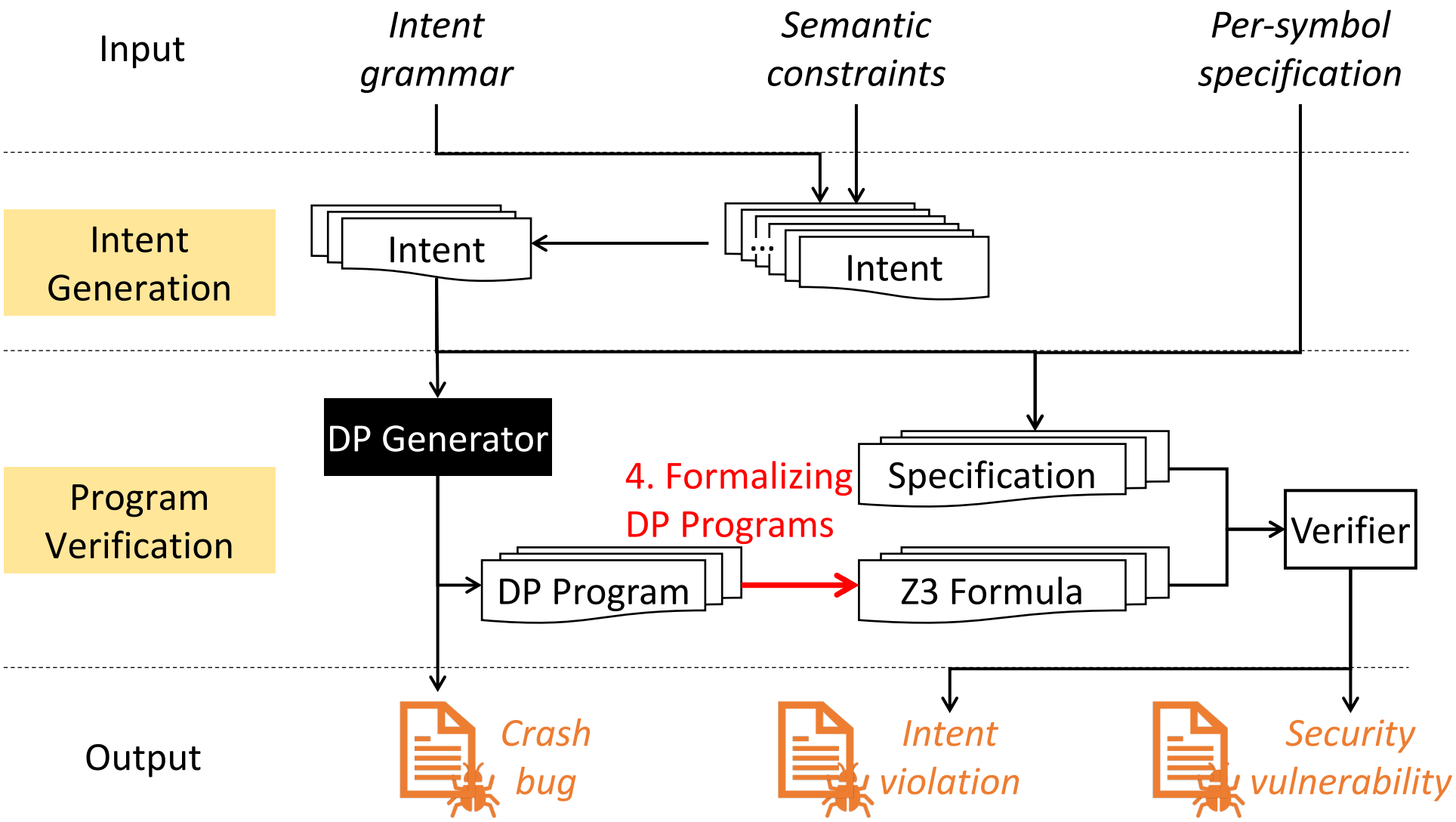
⇨

***count(tp.sport == 80)***

```
def DEC_FUNC:
  counter = 0
```

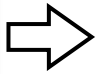
```
def EXEC_FUNC:
  if ((pkt.tcp.sport == 80) == true):
    counter = counter + 1
  return counter
```

# Firebolt Workflow



# Formalizing DP Programs

*Each Programmable Block*
*(parser, ingress, egress, deparser, etc.)*



*Z3 Formulas*

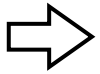
*Example: Match action tables with entries*

```
action a (p) {x = p;}
table t {
      key = k : exact;
      actions = {
      a; no_op;
      }
      default_action = no_op;
}
```

Table entries: 1 => (1)

*Input parameter*

- Match key of table t

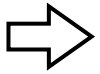


*Free Z3 variables*

- (_ BitVec 32) k

*Output parameter*

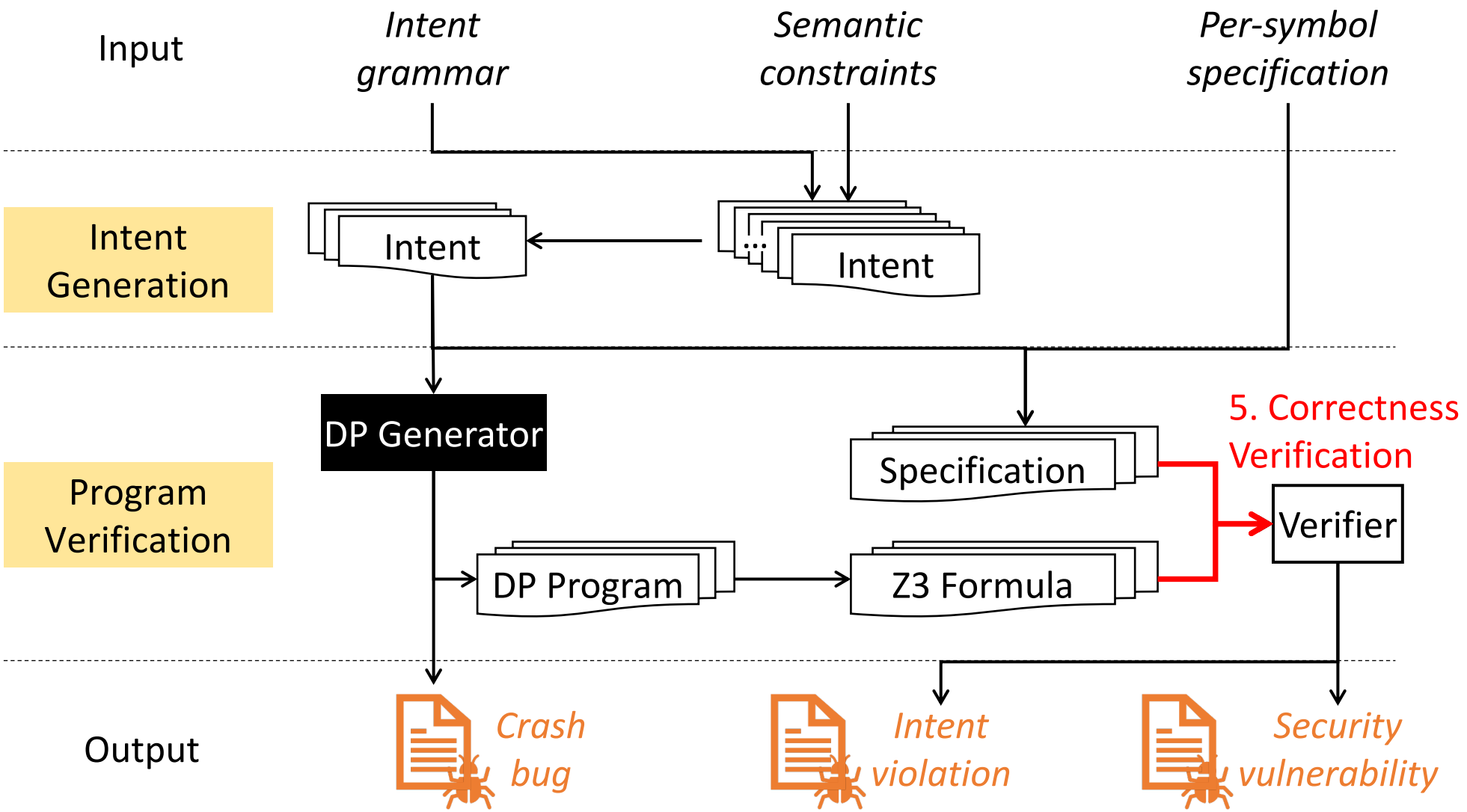
- Variable y



*Output Z3 expression*

- (_ BitVec 32) y = ( if (k=1) 1 else y))

# Firebolt Workflow



# *Evaluation*

**Implementation**

- Ubuntu 16.04 virtual machine with 4GB RAM and two 2.3GHz CPU cores
- ~2000 lines of Python/C++ code (built atop OSDI20-Gauntlet)
- 3 open-source DP generators under test
  - SIGCOMM16-Marple, SIGCOMM17-Sonata, USENIX20-Poise
- 2 DP program verification tools for comparison
  - SIGCOMM21-Aquila, SIGCOMM18-p4v

**Evaluation Goals**

- Bug Coverage
  - How many bugs can Firebolt find?
- Scalability
  - How long does it take to find bugs?
  - How many human efforts does Firebolt take?

# *Bug Coverage*

How many bugs can Firebolt find?

| DP Generator Under Test | # Generated Intents | # Detected Bugs / # Intents Causing Bugs | | |
|---|---|---|---|---|
| | | Crash Bug | Security Vulnerability | Intent Violation |
| **Marple** | 7341 | 1 / 12 | 1 / 7329 | 2 / 23 |
| **Sonata** | 7912 | 0 / 0 | 2 / 7912 | 5 / 243 |
| **Poise** | 2362 | 0 / 0 | 2 / 2362 | 6 / 362 |

- Detect bugs in all three generators
- Detect altogether **5 security vulnerabilities**, **13 intent violations**, **1 crash bug**

# *Scalability*

How many human efforts does Firebolt require? **O(100) LoC**

How long does it take? To debug a DP generator? **< 25 minutes**

| DP Generator Under Test | Human-written LoC | | | Running Time (Total / Average) | |
|---|---|---|---|---|---|
| | Intent Grammar | Semantic Constraints | Per-Symbol Specification | Intent Generation | Program Verification |
| **Marple** | 93 | 70 | 323 | 168s / 23ms | 1204s / 164ms |
| **Sonata** | 34 | 10 | 178 | 27s / 3ms | 926s / 162ms |
| **Poise** | 25 | 25 | 132 | 23s / 10ms | 355s / 150ms |

Can Firebolt save human efforts?

**0.1% to 0.01%**

| DP Generator Under Test | Verifying One Program | Verifying All Programs | Finding All Bugs (1 Bug / 1 Program) |
|---|---|---|---|
| **p4v** | O(1K) | O(1M) | O(10K) |
| **Aquila** | O(100) | O(100K) | O(1K) |
| **Firebolt** | | O(100) | |

# *Conclusion*

- **Firebolt is the first tool designed to debug DP generators**
  - **Thoroughly explore the intent space** to generate syntactically-correct, semantically-valid, and representative intents
  - **Automatically verify DP programs** by formalizing programs and producing specifications
  - **Achieve high bug coverage and high scalability on three DP generators**


Alibaba Cloud | 

Worldwide Cloud Services Partner

*Thanks for your interest in Firebolt*

*cjm21@mails.tsinghua.edu.cn*