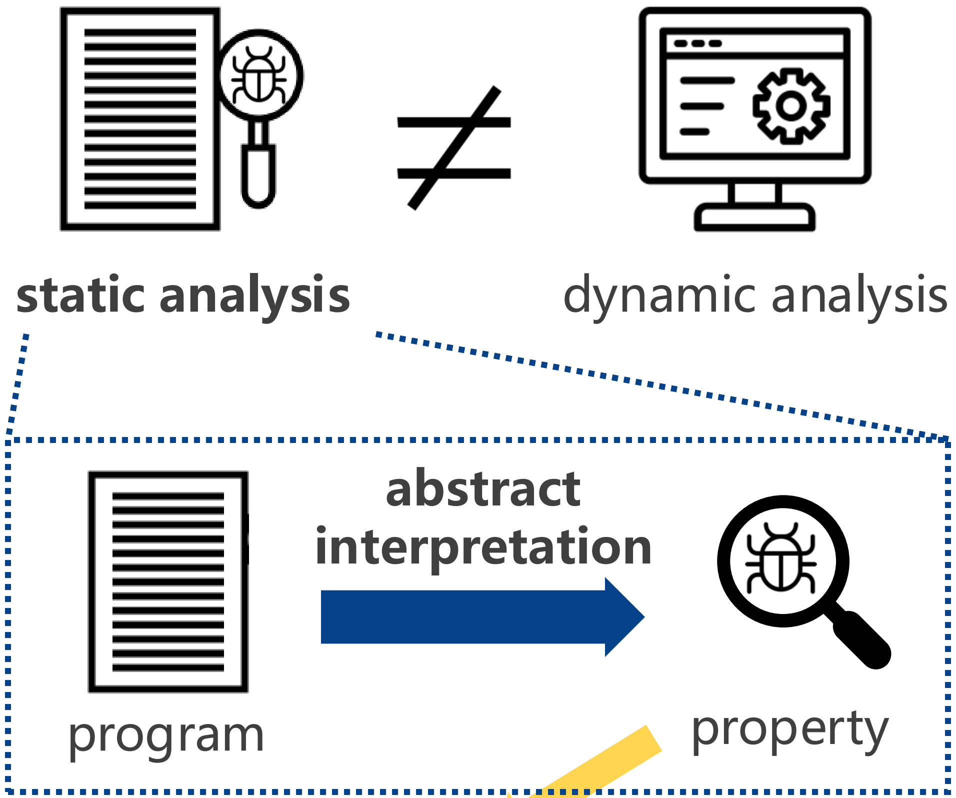
Parf: Adaptive Parameter Refining for Abstract Interpretation

Zhongyi Wang[1], **Linyu Yang**[1], Mingshuai Chen[1], Yixuan Bu[1], Zhiyang Li[1], Qiuye Wang[2], Shengchao Qin[3], Xiao Yi[2], and Jianwei Yin[1].

1: Zhejiang University
2: Fermat Labs, Huawei Inc.
3: Xidian University

Abstract Interpretation-Based Static Analysis



runtime errors (RTEs)

- arithmetic overflows
- division by zero
- null pointer dereference
- ……

```c
void f(void) {
  int i = 0;
  int j = 10;
  while (i < 10) {
    i = i + 1;
    j = j - 1;
  }
}
```

example.c

→ Any overflow here?

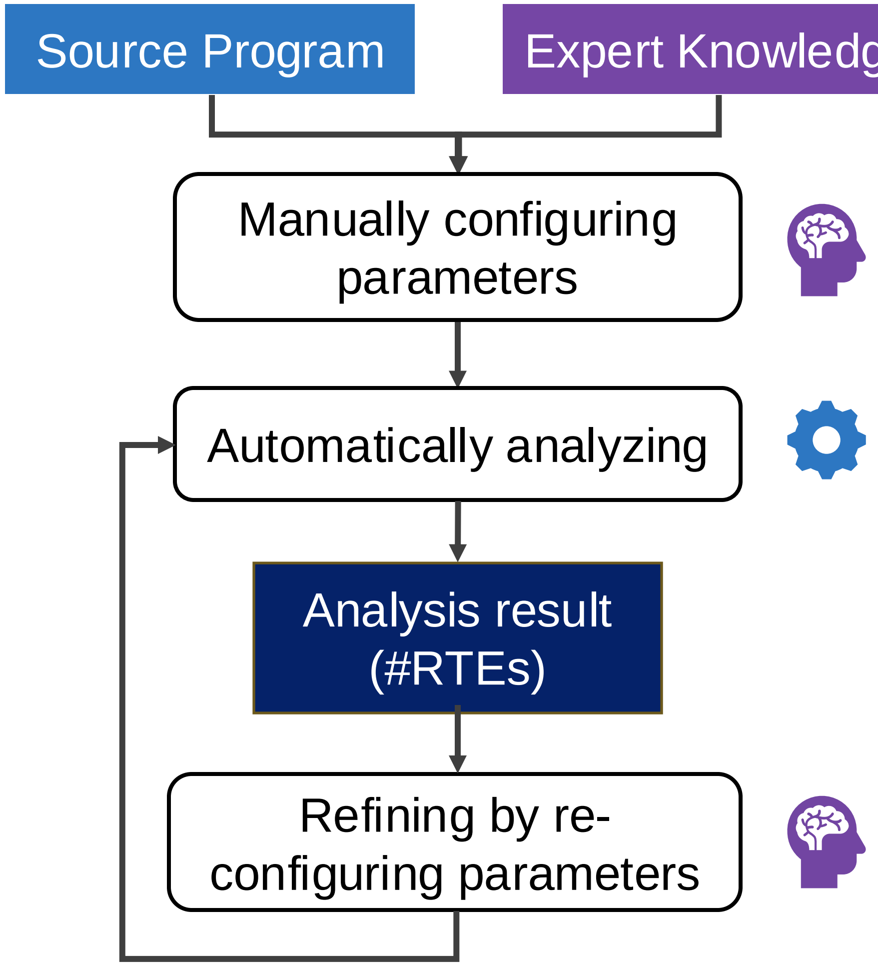
i	j
0	10
1	9
…	…
9	1
10	0

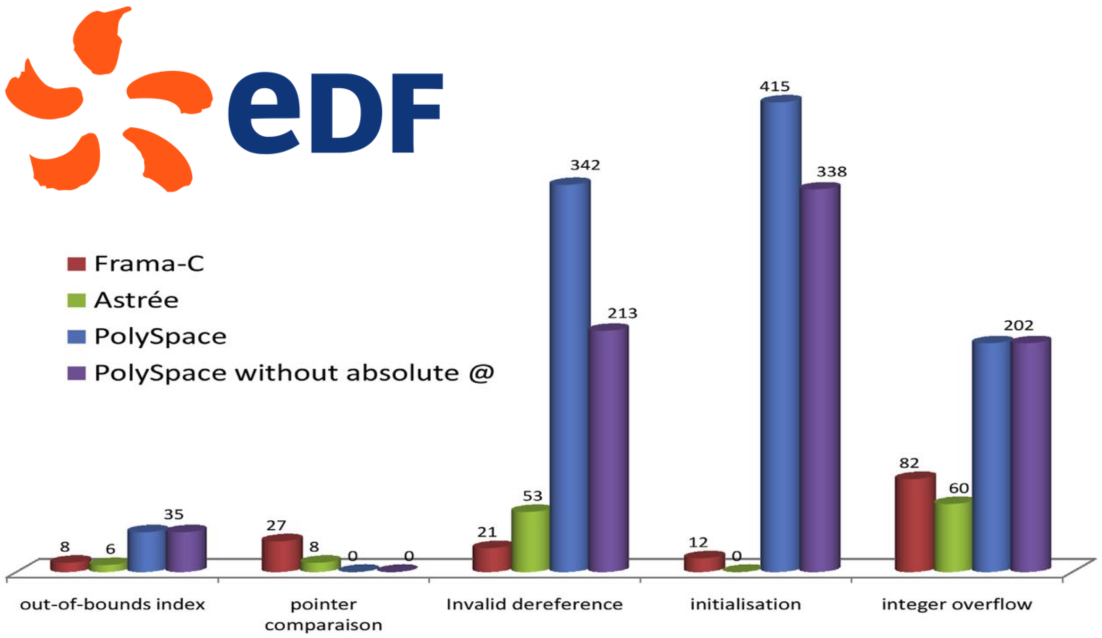
concrete states of i and j

abstract →

i∈[0, 10],
j∈[0, 10]

Workflow of Using Static Analyzer





Ourghanlian A (2015) Evaluation of static analysis tools used to assess software important to nuclear power plant safety. Nucl Eng Technol 47(2):212–218.

An Example of Refining by Reconfiguring Parameters

```c
#include <stdio.h>
int main()
{
  int array[5] = {1, 2, 3, 4, 5};
  int index = 0, sum = 0;

  while (index <= 10) {
    sum += array[index];
    sum *= 2;
    index ++;
  }

  printf("Sum of array: %d\n", sum);
  return 0;
}
```

(a) source C program to be analyzed

```c
#include <stdio.h>
int main(void)
{
  int array[5] = {1, 2, 3, 4, 5}, index = 0, sum = 0;
  while (index <= 10) {
    //@ assert Eva: index_bound: index < 5;
    //@ assert Eva: signed_overflow: sum + array[index] <= 2147483647;
    sum += array[index];
    //@ assert Eva: signed_overflow: sum * 2 <= 2147483647;
    sum *= 2;
    index ++;
  }
  printf("Sum of array: %d\n", sum);
  return 0;
}
```

RTE alarms in the form of ACSL annotation

(b) analysis result with low-precision parameters

```c
#include <stdio.h>
int main(void)
{
  int array[5] = {1, 2, 3, 4, 5};
  int index = 0, sum = 0;

  while (index <= 10) {
    //@ assert Eva: index_bound: index < 5;
    sum += array[index];
    sum *= 2;
    index ++;
  }
  printf("Sum of array: %d\n", sum);
  return 0;
}
```

2 false alarms are eliminated.

(c) analysis result with high-precision parameters

Challenge

Why configuring parameters is tricky and needs expert knowledge?

- a wide range of parameters subject to a huge and possibly infinite joint parameter space

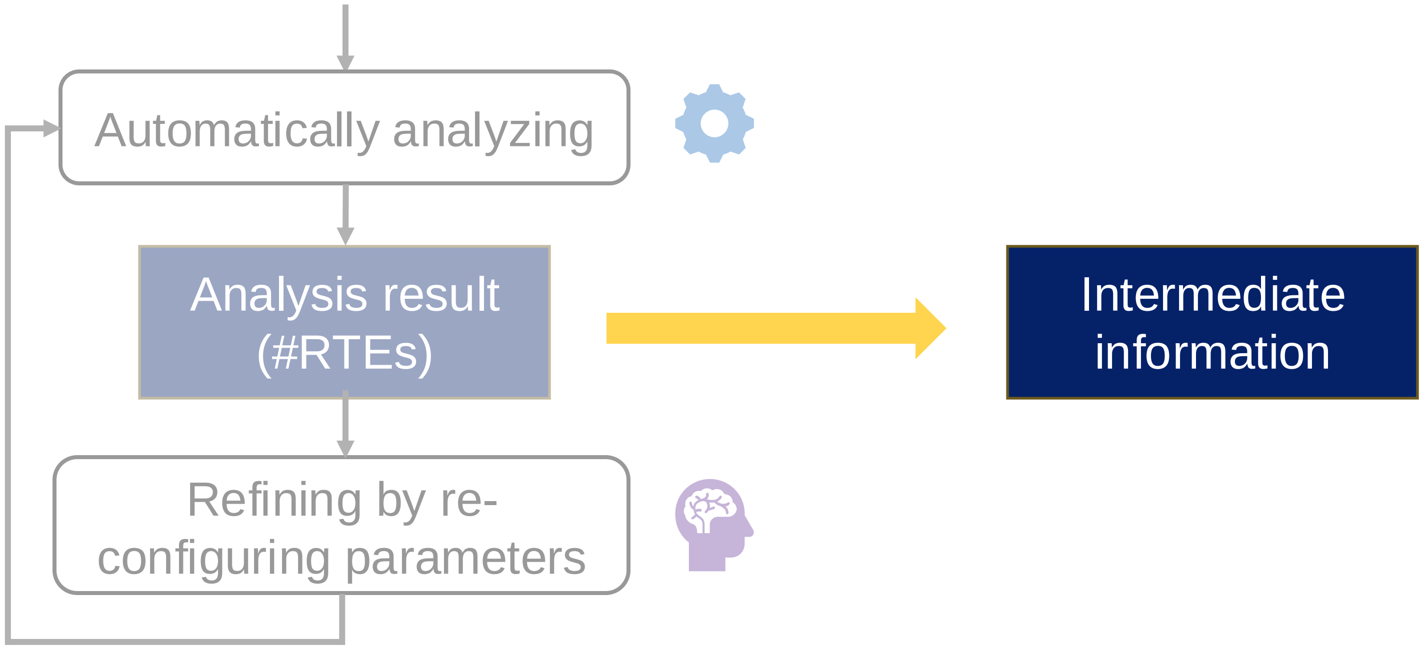
```
[eva] Option -eva-precision 3 detected, automatic configuration of the analysis:
  option -eva-min-loop-unroll set to 0 (default value).
  option -eva-auto-loop-unroll set to 64.
  option -eva-widening-delay set to 2.
  option -eva-partition-history set to 0 (default value).
  option -eva-slevel set to 35.
  option -eva-ilevel set to 24.
  option -eva-plevel set to 70.
  option -eva-subdivide-non-linear set to 60.
  option -eva-remove-redundant-alarms set to true (default value).
  option -eva-domains set to 'cvalue,equality,gauges,symbolic-locations'.
  option -eva-split-return set to '' (default value).
  option -eva-equality-through-calls set to 'none'.
  option -eva-octagon-through-calls set to false (default value).
```

A typical parameter setting of Frama-C/Eva with different parameter types: integer, Boolean, string, and set-of-strings.

Challenge

Why configuring parameters is tricky and needs expert knowledge?

- a wide range of parameters subject to a huge and possibly infinite joint parameter space
- the lack of a framework to utilize intermediate information



Problem Formulation / Parameter Spaces

Type	Sample space	Complete lattice structure
Integer	$PS^1 = \mathbb{Z}^{\infty}_{\geq 0}$	$(\mathbb{Z}^{\infty}_{\geq 0}, \leq, \min, \max)$
Boolean	$PS^2 = \{\mathrm{F}, \mathrm{T}\}$	$(\{F, T\}, \Rightarrow, \wedge, \vee)$
Set-of-strings	$PS^3 = \mathcal{P}(\mathbb{S})$	$(\mathcal{P}(\mathbb{S}), \subseteq, \cap, \cup)$

$$PS \triangleq PS^1 \times PS^2 \times \cdots \times PS^n$$

$$(\mathrm{PS}, \preccurlyeq, \curlywedge, \curlyvee)$$

$$p \curlywedge q \triangleq (p^1 \sqcap q^1, p^2 \sqcap q^2, \ldots, p^n \sqcap q^n),$$

$$p \curlyvee q \triangleq (p^1 \sqcup q^1, p^2 \sqcup q^2, \ldots, p^n \sqcup q^n)$$

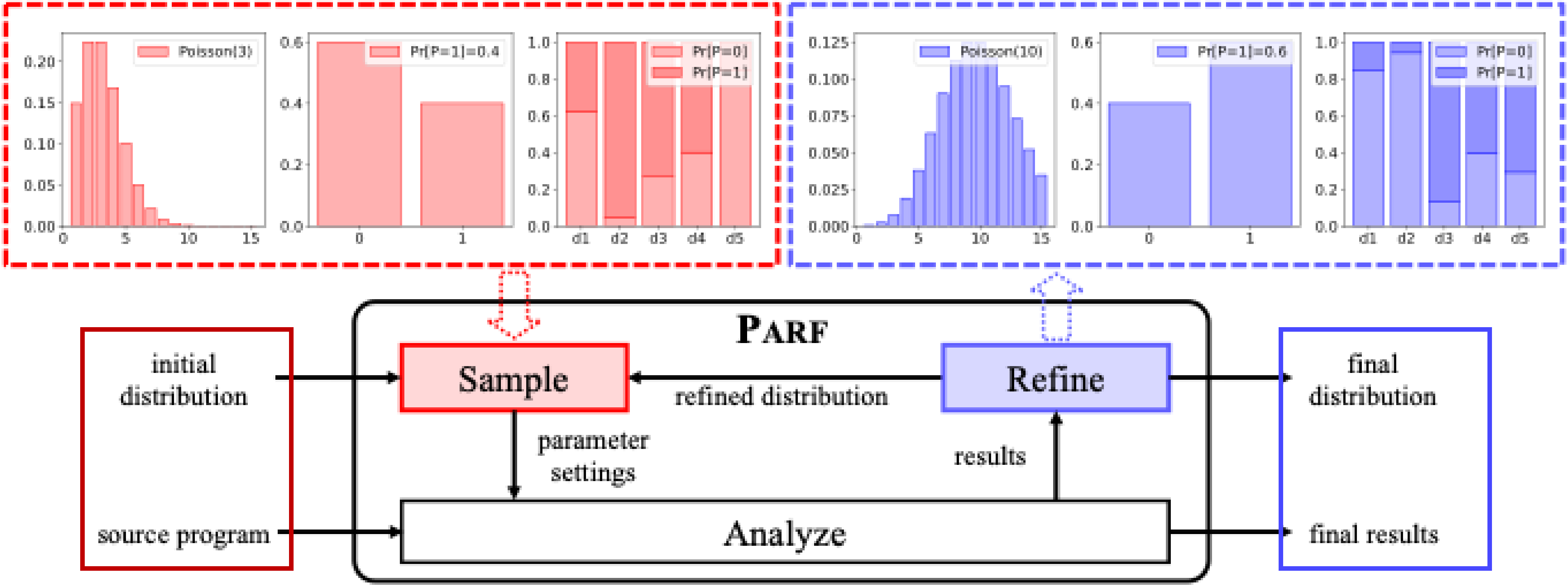
Problem Formulation / Problem Statement

$$\begin{aligned} \mathit{Analyze}: \quad \mathit{Prog} \times \mathit{PS} &\rightarrow \mathcal{P}(A_{\text{uni}}) \\ (\mathit{prog}, p) &\mapsto A_p \end{aligned}$$

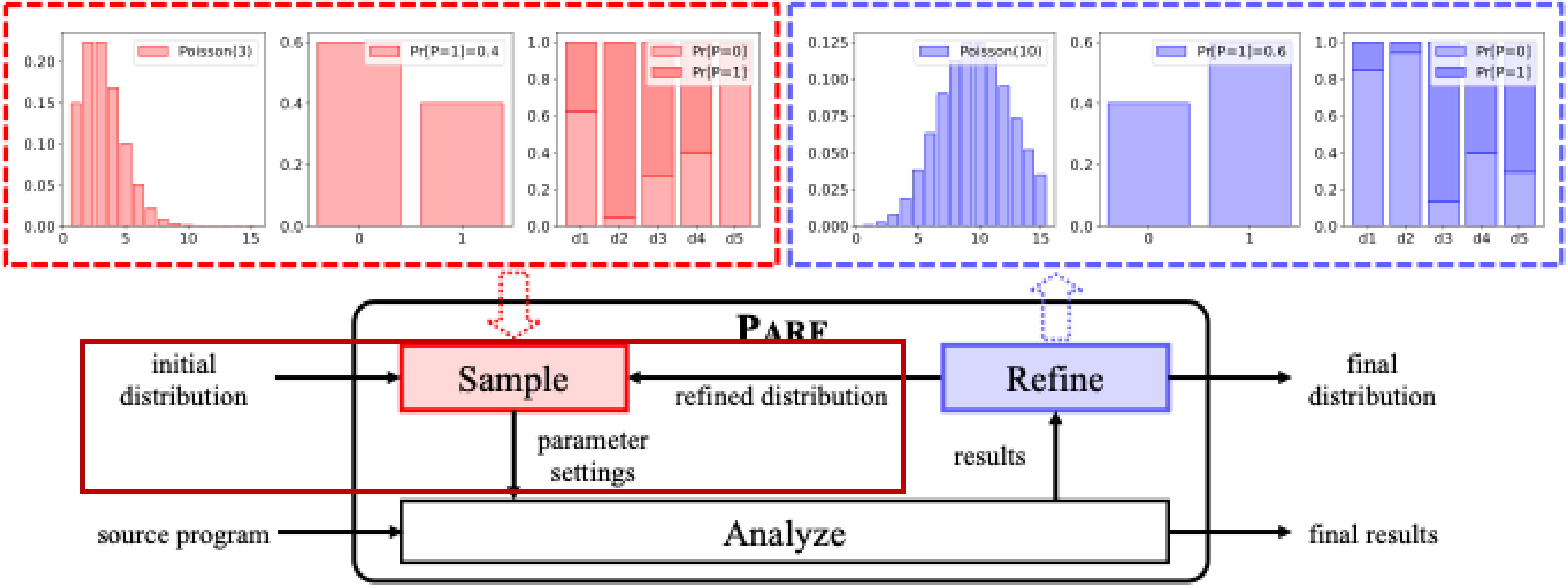
$$p_1 \sqsubseteq p_2 \quad \text{implies} \quad \mathit{Analyze}(\mathit{prog}, p_2) \subseteq \mathit{Analyze}(\mathit{prog}, p_1)$$

Problem Statement. Given a source program $\mathit{prog} \in \mathit{Prog}$, a time budget $T \in \mathbb{R}_{>0}$, an abstraction interpretation-based static analyzer *Analyze*, and the joint space of parameter settings PS of *Analyze*, find a parameter setting $p \in \mathit{PS}$ such that $\mathit{Analyze}(\mathit{prog}, p)$ returns as few alarms as possible within T.

The Parameter Refinement Framework / Overview



The Parameter Refinement Framework / Sample

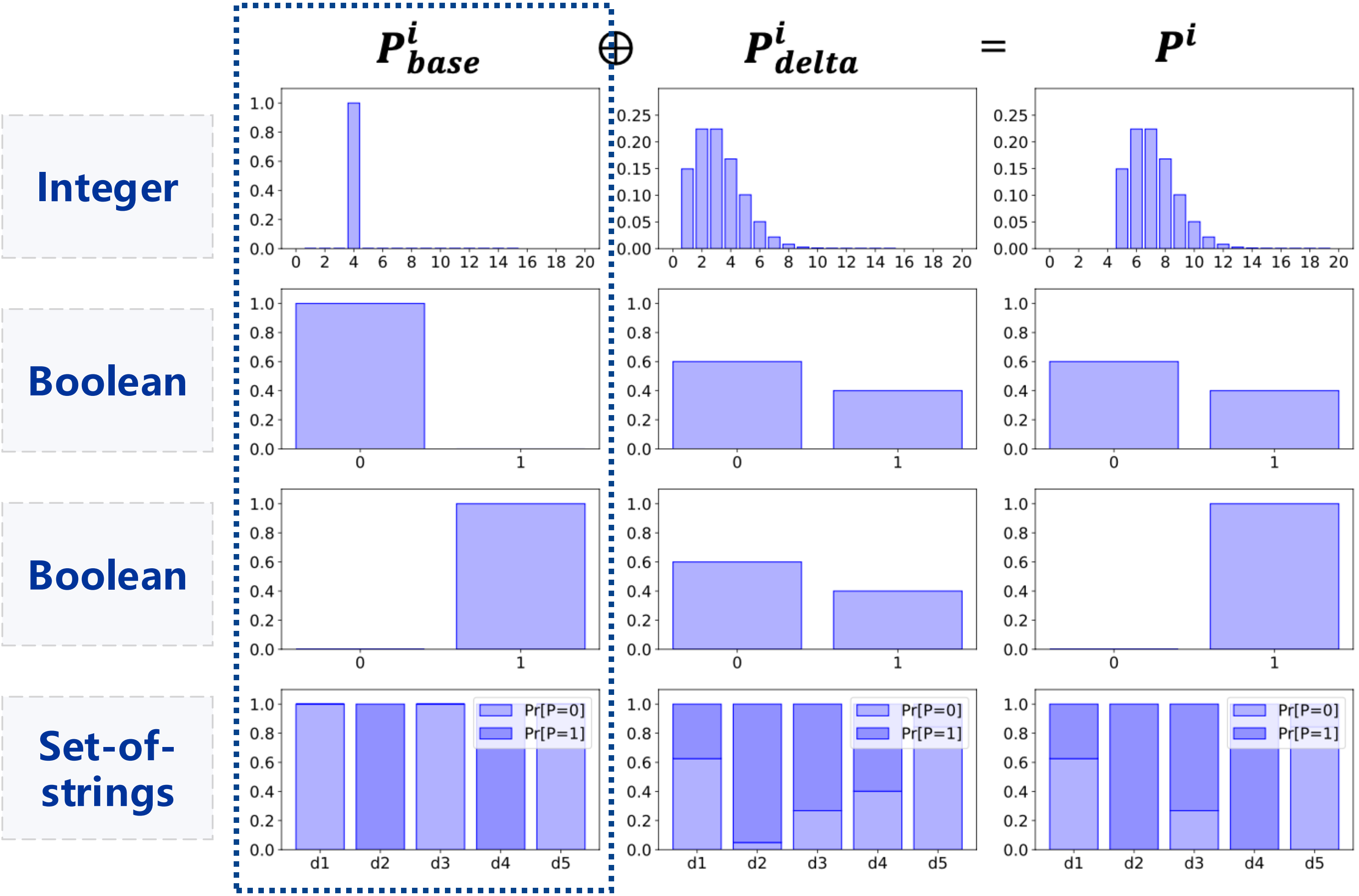


The Parameter Refinement Framework / Sample

$$P^i = \underbrace{P^i_{\text{base}}}_{\text{for retaining}} \oplus \underbrace{P^i_{\text{delta}}}_{\text{for exploring}}$$

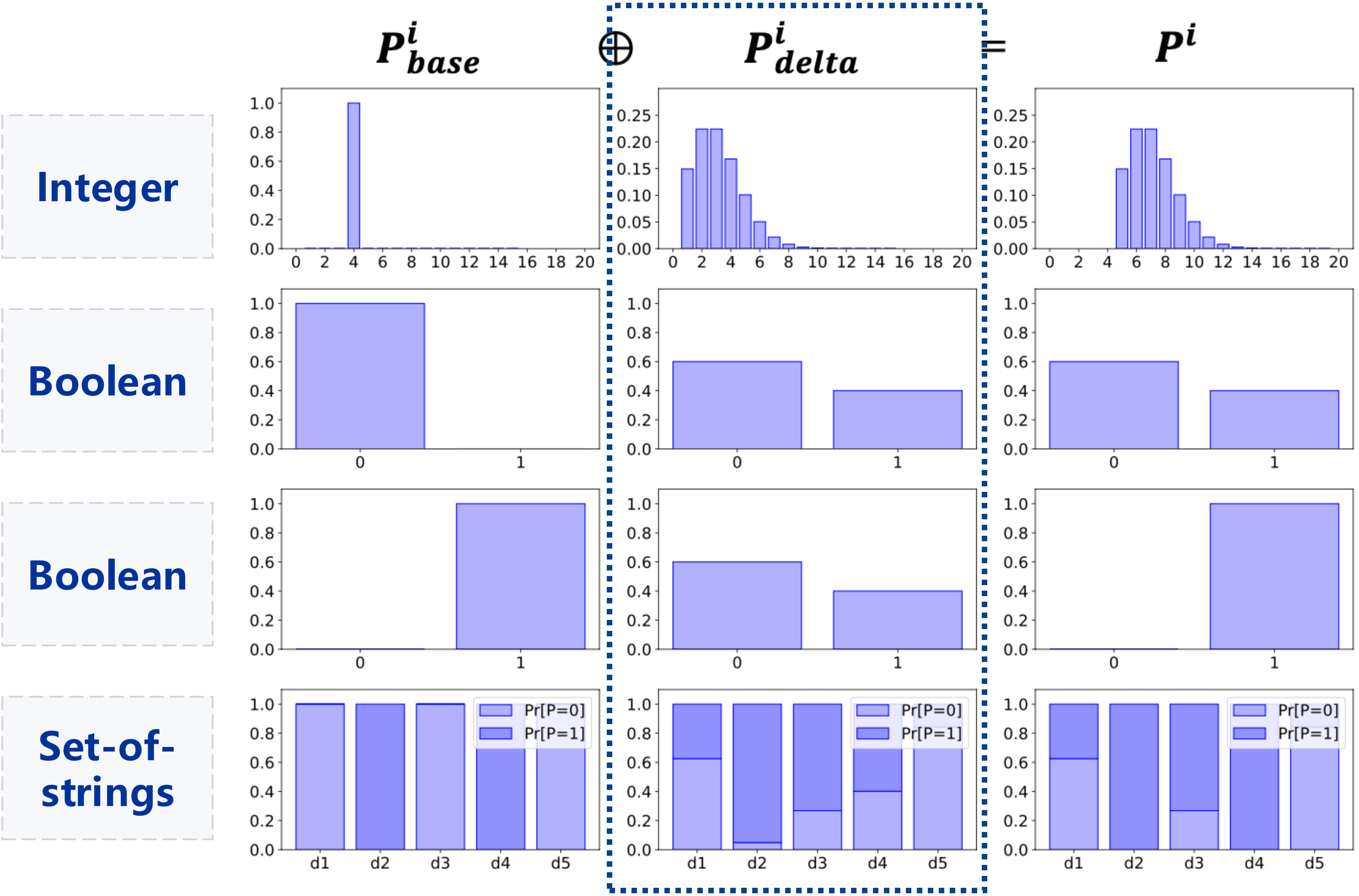
The Parameter Refinement Framework / Sample

$$P^i = \underbrace{P^i_{\text{base}}}_{\text{for retaining}} \oplus \underbrace{P^i_{\text{delta}}}_{\text{for exploring}}$$



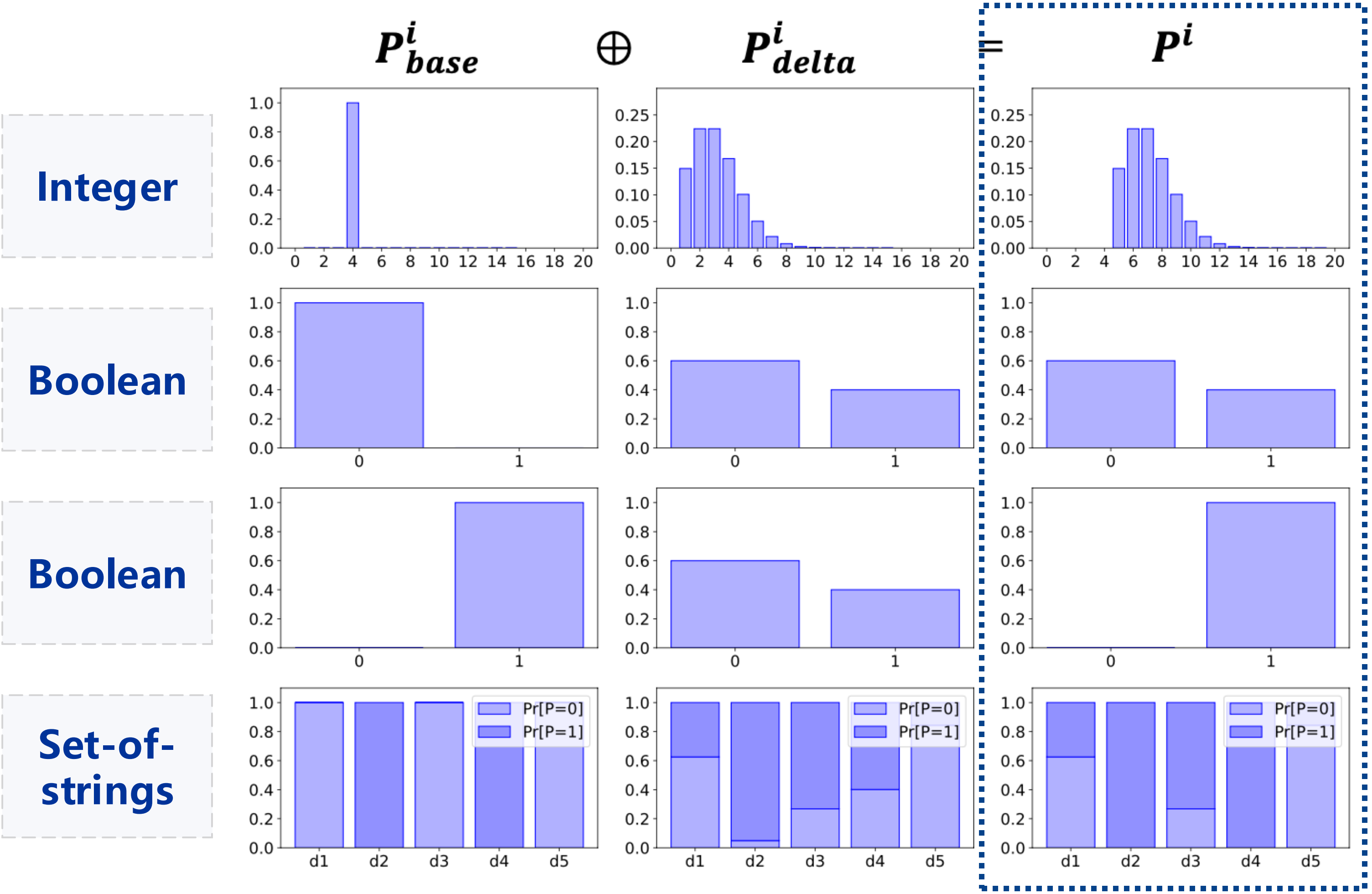
The Parameter Refinement Framework / Sample

$$P^i = \underbrace{P^i_{\text{base}}}_{\text{for retaining}} \oplus \underbrace{P^i_{\text{delta}}}_{\text{for exploring}}$$



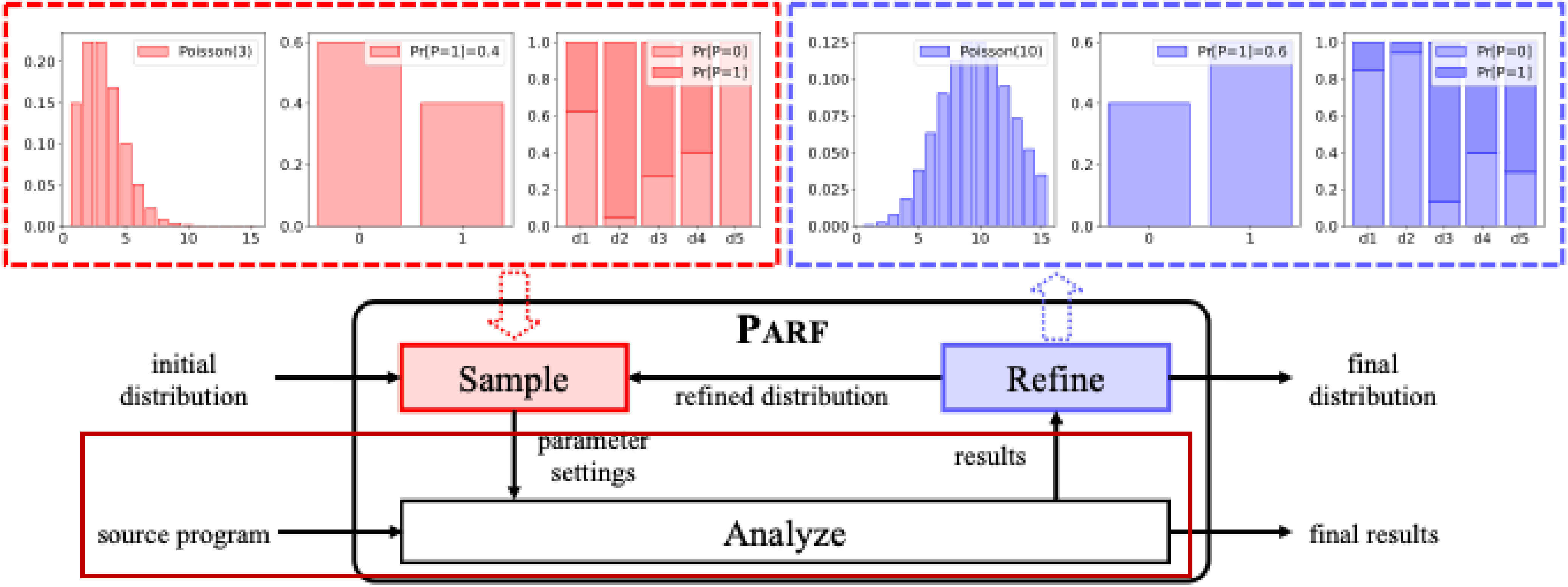
The Parameter Refinement Framework / Sample

$$\underbrace{P^i}_{} = \underbrace{P^i_{\text{base}}}_{\text{for retaining}} \oplus \underbrace{P^i_{\text{delta}}}_{\text{for exploring}}$$

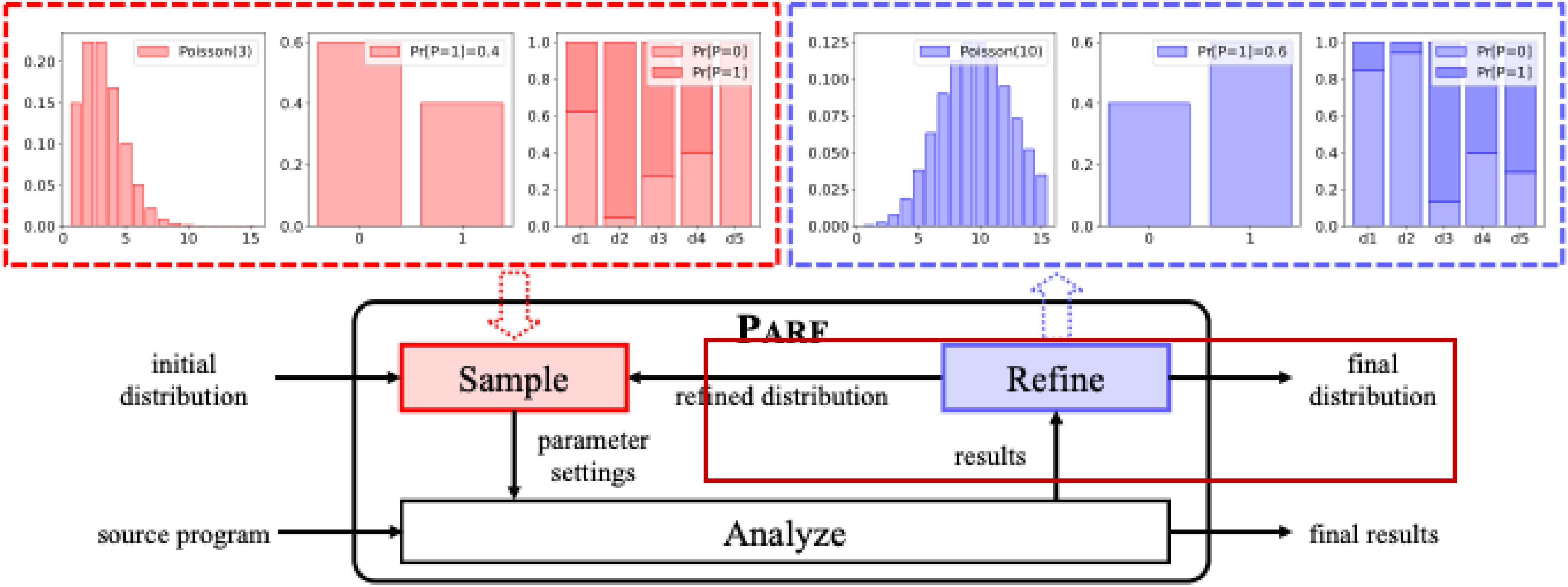


$$P = P_{\text{base}} \oplus P_{\text{delta}} \triangleq \left(P^1_{\text{base}} \oplus P^1_{\text{delta}}, \ldots, P^n_{\text{base}} \oplus P^n_{\text{delta}}\right)$$

The Parameter Refinement Framework / Sample



The Parameter Refinement Framework / Sample



The Parameter Refinement Framework / Refine

Algorithm 2 `Refine`: Incremental Refining

Input: List of parameter settings p_list, list of results R_list, universe alarms A_{uni}, and P_{base}, P_{delta}.

Output: Refined distributions P'_{base} and P'_{delta}.

```
/* Step 1: Refine P_base */
P'_base ← P_base ;
for all a ∈ A_uni do
    P_a ← ⊤ ;
    for all ⟨p, A⟩ ∈ R_list and a ∉ A do
        p_a ← p_a ⊓ p ;
    end for
    if p_a ≠ ⊤ then
        P'_base ← P'_base ⊔ p_a ;
    end if
end for

/* Step 2: Refine P_delta */
η_scale ← (2×|R_list|+1) / |p_list| ;
P'_delta ← η_scale ⊗ P_delta ;
return P_base, P_delta ;
```

$\langle p, A \rangle$

the sampled parameter setting

the corresponding set of alarms

$$P'_{base} = \bigsqcup_{a \in A_{uni}} p_a = \bigsqcup_{a \in A_{uni}} \left(\bigsqcap_{\substack{\langle p,A \rangle \in R\_list \\ a \notin A}} p \right)$$

$$\eta_{scale} \otimes P_{delta} = \left(\eta_{scale} \otimes P^1_{delta}, \cdots, \eta_{scale} \otimes P^n_{delta} \right)$$

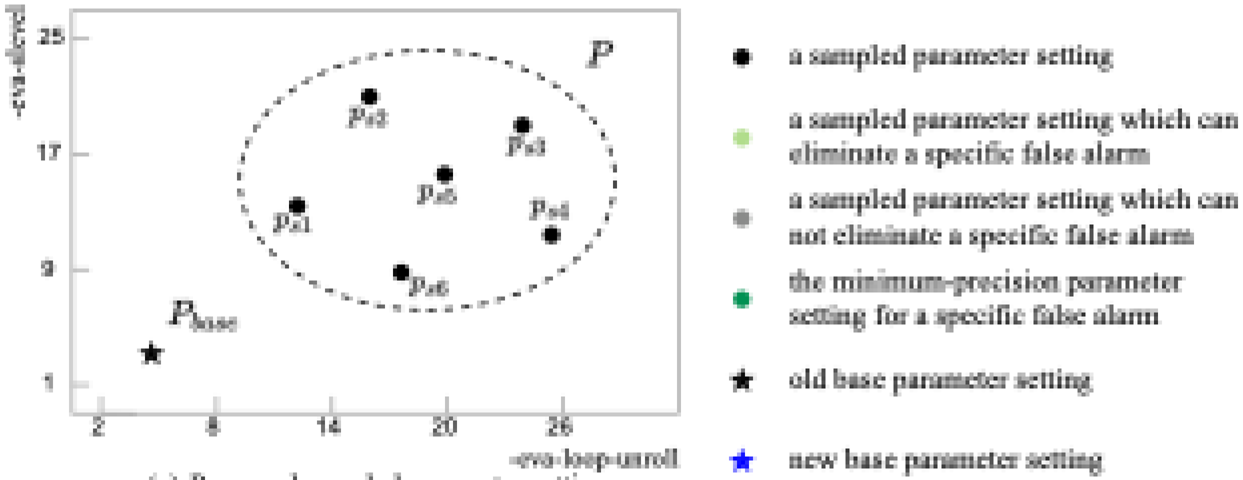
The Parameter Refinement Framework / Refine P_{base}

$$P'_{\text{base}} = \bigsqcup_{a \in A_{\text{uni}}} p_a = \bigsqcup_{a \in A_{\text{uni}}} \left(\bigsqcap_{\substack{\langle p, A \rangle \in R\_list \\ a \notin A}} p \right)$$

the "parameter setting with lowest precision" P'_{base} eliminating all newly found false alarms

the "parameter setting with lowest precision" p_a eliminating a

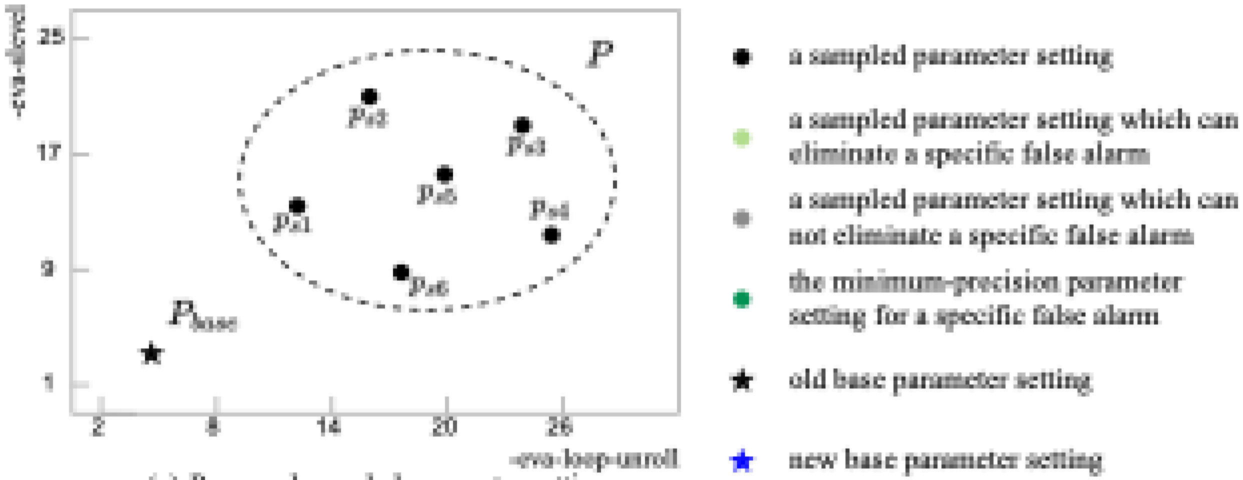
The Parameter Refinement Framework / Case Study



(a) P_{base} and sampled parameter settings

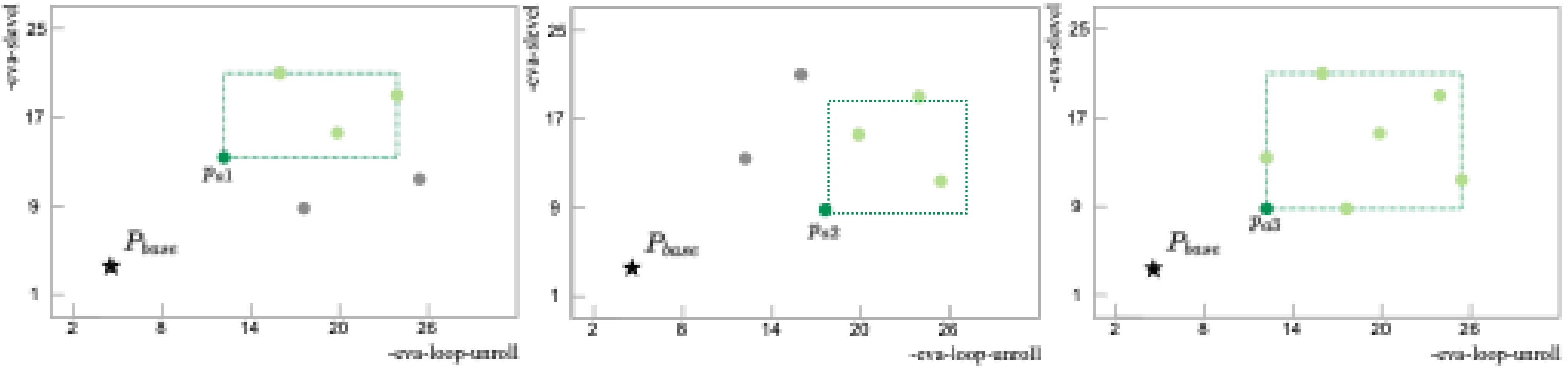
$$P'_{\text{base}} = \bigsqcup_{a \in A_{\text{uni}}} p_a = \bigsqcup_{a \in A_{\text{uni}}} \left(\bigsqcap_{\substack{\langle p, A \rangle \in R\_list \\ a \notin A}} p \right)$$

The Parameter Refinement Framework / Case Study



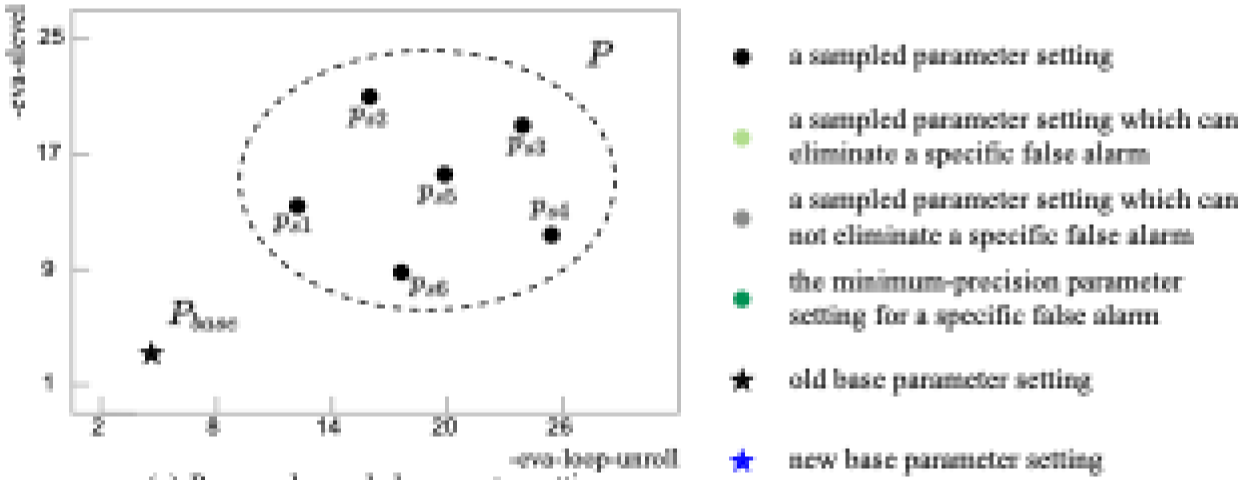
(a) P_{base} and sampled parameter settings

$$P'_{\text{base}} = \bigsqcup_{a \in A_{\text{uni}}} p_a = \bigsqcup_{a \in A_{\text{uni}}} \left(\bigsqcap_{\substack{\langle p, A \rangle \in R\_list \\ a \notin A}} p \right)$$



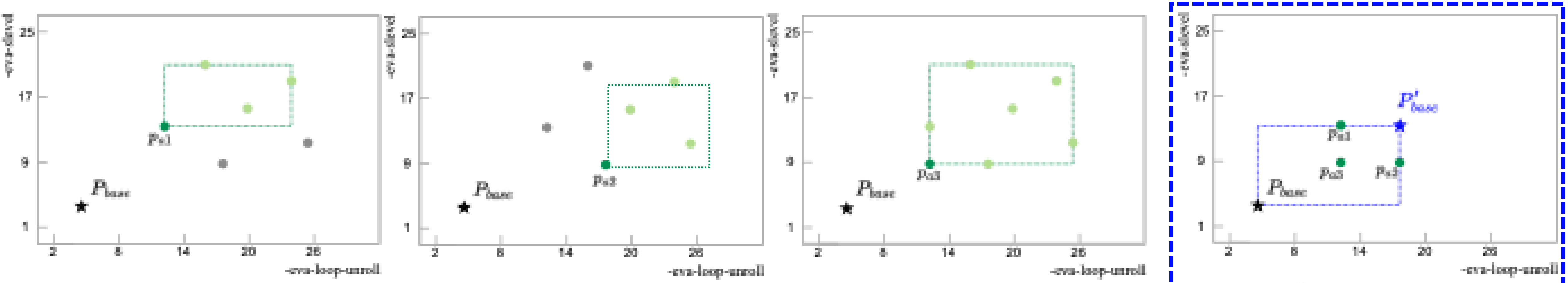
(b) $p_{a1} = p_{s1} \sqcap p_{s2} \sqcap p_{s3} \sqcap p_{s5}$ (c) $p_{a2} = p_{s3} \sqcap p_{s4} \sqcap p_{s5} \sqcap p_{s6}$ (d) $p_{a3} = p_{s1} \sqcap p_{s2} \sqcap p_{s3} \sqcap p_{s4} \sqcap p_{s5} \sqcap p_{s6}$

The Parameter Refinement Framework / Case Study



(a) P_{base} and sampled parameter settings

$$P'_{base} = \bigsqcup_{a \in A_{uni}} p_a = \bigsqcup_{a \in A_{uni}} \left(\bigsqcap_{\substack{\langle p, A \rangle \in R\_list \\ a \notin A}} p \right)$$



(b) $p_{a1} = p_{s1} \sqcap p_{s2} \sqcap p_{s3} \sqcap p_{s5}$ (c) $p_{a2} = p_{s3} \sqcap p_{s4} \sqcap p_{s5} \sqcap p_{s6}$ (d) $p_{a3} = p_{s1} \sqcap p_{s2} \sqcap p_{s3} \sqcap p_{s4} \sqcap p_{s5} \sqcap p_{s6}$ (e) $P'_{base} = P_{base} \sqcup p_{a1} \sqcup p_{a2} \sqcup p_{a3}$

Evaluation: Research Questions

RQ1: How does Parf compare against other parameter-selecting strategies?
RQ2: How does Parf perform on different hyper-parameters?
RQ3: Can Parf be generalized to other static analyzers?
RQ4: Can Parf improve Frama-C in verification competitions?

Experiment Settings

Experiment environments:

- RQ1, RQ2, and RQ4: 8-core Apple M2 processor,16GB RAM, 64-bit macOS Sonoma 14
- RQ3: 16-core Intel i7 processor, 16GB RAM, Arch Linux

Benchmarks:

- RQ1, RQ2 , and RQ3: Frama-C official Open Source Case Study (OSCS) benchmarks
- RQ4: verification tasks of SV-COMP 2022, the NoOverflows category with a specific version called Frama-C-SV

Baselines:

- Default: default parameter settings of Frama-C/Eva or Mopsa
- Official: official parameter settings provided by Frama-C together with the OSCS benchmarks
- Expert: dynamic parameter-tuning strategy for Frama-C/Eva, sequentially increases the parameters from -eva-precision 0 to -eva-precision 11 for analysis until the given time budget is exhausted or the highest precision level is reached

Time Budget:

- 1 hour for each benchmark

Evaluation: RQ1

OSCS Benchmark Details				#Alarms (the fewer, the more accurate)			
Benchmark name	LOC	#statements	-eva-precision	Default	Expert	Official	Parf
gzip124	8166	4835	0	884	885	866	**810**
miniz-ex1	10844	3659	1	2291	1832	2291	1828
miniz-ex2	10844	5589	1	2742	2220	2742	**2172**
miniz-ex3	10844	3747	1	577	552	577	**442**
miniz-ex5	10844	3430	1	425	402	425	**377**
miniz-ex6	10844	2073	1	220	198	220	**173**
monocypher	25263	4126	1	606	570	568	606
debie1	8972	3243	2	33	3	**1**	19
kilo	1276	1078	2	523	445	688	**429**
x509-parser	9457	3112	3	208	198	198	**187**
miniz-ex4	10844	1246	4	258	217	258	**189**
tsvc	5610	5478	4	413	355	379	356
2048	440	329	6	7	5	7	**4**
libspng	4455	2377	6	186	122	122	**113**
microstrain	51007	3216	6	1177	616	646	**598**
mini-gmp	11706	628	6	83	71	83	71
safestringlib	29271	13029	6	855	**256**	300	356
stmr	781	500	6	63	58	59	58
qlz-ex3	1168	294	8	94	82	94	**75**
semver	1532	728	9	29	22	25	22
genann	1183	1042	9	236	69	77	69
kgflags-ex2	1455	736	10	33	19	33	19
chrony	37177	41	11	9	7	8	7
hiredis	7459	87	11	9	0	9	0
icpc	1302	424	11	9	1	1	1
jsmn-ex1	1016	1219	11	58	1	1	1
jsmn-ex2	1016	311	11	68	1	1	1
kgflags-ex1	1455	474	11	11	0	11	0
khash	1016	206	11	14	2	14	2
line-following-robot	6739	857	11	1	1	1	1
papabench	12254	36	11	1	1	1	1
qlz-ex1	1168	229	11	68	11	68	11
qlz-ex2	1168	75	11	8	8	8	8
qlz-ex4	1168	164	11	17	13	17	13
solitaire	338	396	11	216	18	213	18
tutorials	325	89	11	5	1	5	**0**
tweetnacl-usable	1204	659	11	126	25	30	25
Overall (tied-best+**exclusively best**)				3/37	23/37	8/37	34/37 (91.9%)
Overall (**exclusively best**)				0/37	1/37	1/37	12/37 (32.4%)

RQ1: How does Parf compare against other parameter-selecting strategies?

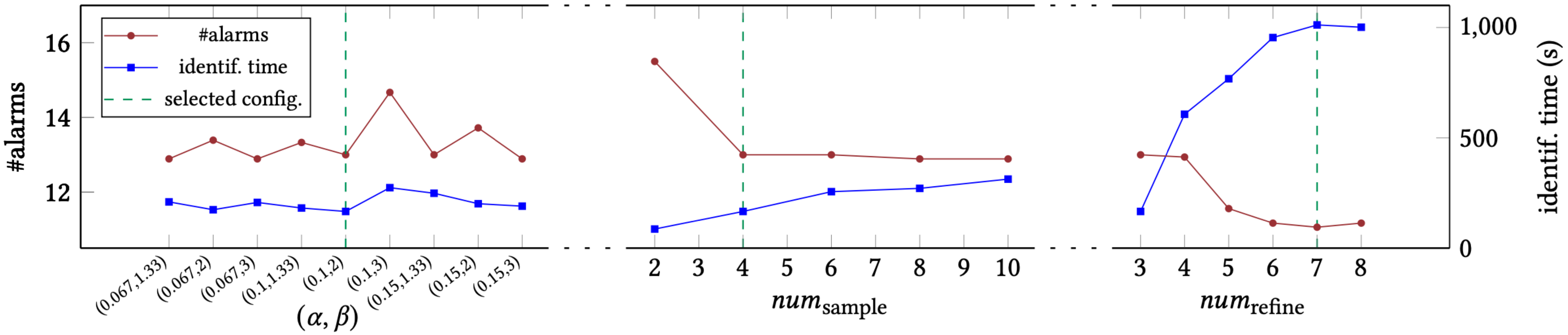
Parf achieves the **best** results on **91.9%** (34/37) benchmarks, and **exclusively best** on **32.4%** (12/37) benchmarks.

Parf performs almost the same as the expert strategy in programs with low analysis complexity (-eva-precision>=9).

Parf achieves **exclusively best** on **57.9%** (11/19) on programs with **high analysis complexity** (-eva-precision<9).

Evaluation: RQ2

RQ2: How does Parf perform on different hyper-parameters?



Evaluation: RQ3

OSCS Benchmark Details				#Alarms of Mopsa (RQ3)	
Benchmark name	LOC	#statements	-eva-precision	Default	Parf
2048	440	329	6	141	**67**
chrony	37177	41	11	–	–
debie1	8972	3243	2	8245	**5656**
genann	1183	1042	9	1308	1308
gzip124	8166	4835	1	–	–
hiredis	7459	87	11	43	43
icpc	1302	424	11	11	**10**
jsmn-ex1	1016	1219	11	1762	**1253**
jsmn-ex2	1016	311	11	87	**86**
kgflags-ex1	1455	474	11	280	280
kgflags-ex2	1455	736	10	386	386
khash	1016	206	11	19	19
kilo	1276	1078	2	5299	5290
libspng	4455	2377	6	–	–
line-following-robot	6739	857	11	–	–
microstrain	51007	3216	6	6237	6196
mini-gmp	11706	628	6	513	**491**
miniz-ex1	10844	3659	1	3020	3004
miniz-ex2	10844	5589	1	3916	3899
miniz-ex3	10844	3747	1	2808	2792
miniz-ex4	10844	1246	4	162	162
miniz-ex5	10844	3430	1	1575	**1474**
miniz-ex6	10844	2073	1	1197	**1075**
monocypher	25263	4126	1	TO	TO
papabench	12254	36	11	–	–
qlz-ex1	1168	229	11	82	82
qlz-ex2	1168	75	11	50	50
qlz-ex3	1168	294	8	–	–
qlz-ex4	1168	164	11	–	–
safestringlib	29271	13029	6	–	–
semver	1532	728	9	3556	**2850**
solitaire	338	396	11	700	**663**
stmr	781	500	6	1391	1391
tsvc	5610	5478	4	–	–
tutorials	325	89	11	–	–
tweetnacl-usable	1204	659	11	667	**657**
x509-parser	9457	3112	3	364	**339**
Overall (tied-best+**exclusively best**)				14/27 (51.9%)	26/27 (96.3%)
Overall (**exclusively best**)				0/27 (0.0%)	12/27 (44.4%)

RQ3: Can Parf be generalized to other static analyzers?

Matthieu Journault et al. 2019. Combinations of Reusable Abstract Domains for a Multilingual Static Analyzer. In VSTTE (Lecture Notes in Computer Science, Vol. 12031). Springer, 1–18.

Evaluation: RQ4

RQ4: Can Parf improve Frama-C in verification competitions?

Setting	Verification Result				Score
	correct	wrong	unknown	failure	
FRAMA-C-SV$_{\text{precision11}}$	146	3	272	33	186
FRAMA-C-SV$_{\text{PARF}}$	**151**	3	300	**0**	**196**

Summary

A new framework for adaptively tuning external parameters of abstract interpretation-based static analyzers, which is particularly practical for large-scale programs.