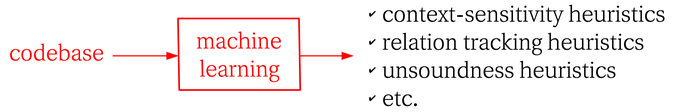
Automatically Generating Features for Learning Program Analysis Heuristics for C-like Languages

Kwonsoo Chae (KU), Hakjoo Oh (KU), Kihong Heo (SNU), Hongseok Yang (Oxford)

1. Problem

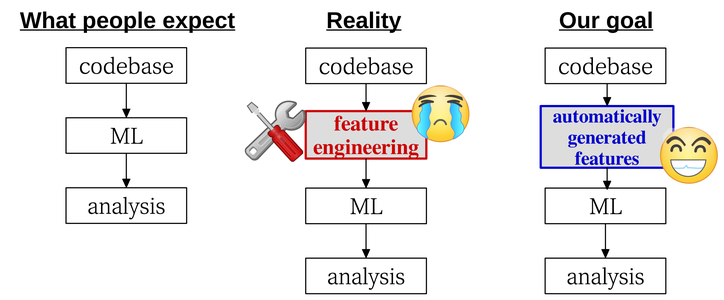
- Data-driven static analysis



- Common limitations of existing data-driven approaches

  → manual feature engineering:

✖ Time-comsuming ✖ Domain knowledge ✖ Analysis-dependent



2. Learning Analysis Heuristics

- Example: partially flow-sensitive interval analysis

```
1  x = 0;  y = 0;  z = input ();  w = 0;
2  y = x;  y++;
3  assert (y > 0);     // Query 1   Apply FS!
4  assert (z > 0);     // Query 2   FI
5  assert (w == 0); // Query 3   FI
```

- Learn a classifier $\mathcal{C} : \mathbb{B}^k \to \mathbb{B}$ that selects **Query 1** only. Then,
  (1) Collect queries that $\mathcal{C}$ predicts positively.
  (2) Collect program parts that are required to prove the queries.

Our goal: Automatically generate features to automatically represent each query as a feature vector ($\mathbb{B}^k$).

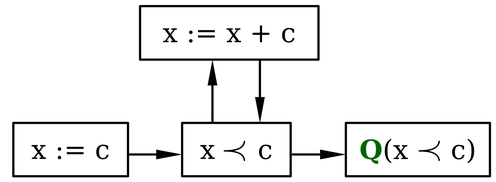
3. Automatic Feature Generation

```
1  a = 0;
2  while (1) {
3      b = unknown ();
4      if (a > b)
5          if (a < 3)
6              assert (a < 5);
7      a++;
8  }
```

(a) Original program →

```
1  a = 0;
2  while (1) {
3      if (a < 3)
4          assert (a < 5);
5      a++;
6  }
```

(b) Feature program →



(c) Abstract data flow graph (Feature)

reduce : $\mathbb{P} \times (\mathbb{P} \to \mathbb{B}) \to \mathbb{P}$

- Synthesize a small program that describes the key aspect of FS-effectiveness using a program reducer.

- Generalize the program text (commands are abstracted).
- Proper abstraction level ?: generalization vs. preservation
  → search via iterative cross validation

4. Results

- Query prediction & final analysis

Inst-ance	Query prediction		Analysis		Manual	
	Precision	Recall	Prove	Cost	Prove	Cost
ITV	74.5 %	75.8 %	**80.2 %**	**2.0x**	55.1 %	2.3x
PTR	76.7 %	77.4 %	**81.2 %**	**2.4x**	none	
REL	79.0 %	79.9 %	**81.1 %**	**1.4x**	96.2 %	1.8 x

- Generated feature programs
  (Top 2 in partially-FS interval analysis)

Feature program 1

```
int buf [10];
for (i = 0; i < 7; i++) {
    buf[i] = 0;  // Query 1
}
```

Feature program 2

```
k = 255; p = malloc (k);
while (k > 0) {
  *(p + k) = 0; // Query 2
  k - - ;
}
```

5. Takeaways

- **A framework for automatically generating features** for learning program analysis heuristics
- A method that uses **a program reducer** for generating feature programs, which capture important behaviors of static analysis
- The notion of **abstract data flow graphs** as **generic features** in data-driven static anaylsis
- Experimental **evaluations with three different kinds of static analyses**.

6. Conclusion

- We have identified and solved the problem of manual feature engineering in data-driven static analysis.