Automatic Predicate Abstraction of C Programs

Presented by Xuankang Lin

Outline

- **Main contribution**
- Introduction to C2BP
- Challenges of Predicate Abstraction in C
- Conclusion

Main Contribution

- Model checkers typically operate on abstractions of systems.
- Use predicate abstraction to model check real softwares.
- The first to apply Predicate Abstraction to real world programming languages (C).

Outline

- Main contribution
- **Introduction to C2BP**
- Challenges of Predicate Abstraction in C
- Conclusion

C2BP - Demo

```
typedef struct cell {
  int val;
  struct cell* next;
} *list;

list partition(list *l, int v) {
  list curr, prev, newl, nextCurr;

  curr = *l;
  prev = NULL;
  newl = NULL;
  while (curr != NULL) {
    nextCurr = curr->next;
    if (curr->val > v) {
      if (prev != NULL) {
        prev->next = nextCurr;
      }
      if (curr == *l) {
        *l = nextCurr;
      }
      curr->next = newl;
L:    newl = curr;
    } else {
```

```
void partition() {
  bool {curr==NULL},  {prev==NULL};
  bool {curr->val>v}, {prev->val>v};
  {curr==NULL}  = unknown();
  {curr->val>v} = unknown();
  {prev==NULL} = true;
  {prev->val>v} = unknown();
  skip;
  while(*) {
    assume(!{curr==NULL});
    skip;
    if (*) {
      assume({curr->val>v});
      if (*) {
        assume(!{prev==NULL});
        skip;
      }
      if (*) {
        skip;
      }
      skip;
L:    skip;
    } else {
```

```
// curr = *l;

// prev = NULL;

// newl = NULL;
// while(curr!=NULL)
//
//  nextCurr = curr->next
//  if (curr->val > v) {
//
//    if (prev != NULL) {
//
//     prev->next = nextCurr;
//    }
//    if (curr == *l) {
//     *l = nextCurr;
//    }
//    curr->next = newl;
//    newl = curr
//  } else {
```

C2BP

- Given a C program P and a set E = $\{\phi_1, \phi_2, \ldots, \phi_n\}$ of predicates, C2BP automatically constructs an abstraction of P, i.e. a boolean program BP(P,E).
- BP(P, E) is a program that has identical control structure to P but contains only |E| boolean variables.
- “Abstraction”: the set of execution traces of BP(P,E) is a superset of the set of execution traces of P.
- Soundness: a path in P => a path in BP(P, E)

After C2BP

- BP(P, E) can be analyzed precisely using a BEBOP that performs inter-procedural data-flow analysis using binary decision diagrams.
- BEBOP is a symbolic model checker for boolean programs.
- BEBOP can generate an invariant representing the reachable states at a program point of the boolean program.
- This invariant can be useful, e.g. to refine alias information.

Outline

- Main contribution
- Introduction to C2BP
- **Challenges of Predicate Abstraction in C**
- Conclusion

Challenges of Predicate Abstraction in C

- **Pointers**
- Procedures & Procedure Calls
- Unknown Values
- Precision-efficiency tradeoff

Challenge - Pointers & Aliasing

- Use weakest liberal precondition to propagate. WP(op, Q)
- "weakest": ∀ P . {P} op {Q}, P => WP(op, Q)
- Problem: { Q[e/x] } x := e { Q } does not hold with pointers!
  - e.g. WP(x := 3, *p > 5) is not *p > 5. Because p may points to x.

Challenge - Pointers & Aliasing

- Solution: divide into two cases, when there is aliasing & when there isn't.

- For WP(x:=e, ϕ) where y is a pointer mentioned in ϕ

  - $\phi[x, e, y] = (\&x = \&y \land \phi[e/y]) \lor (\&x \neq \&y \land \phi)$

- Constraint on C program: no multiple dereference (e.g. **p)

Challenge - Pointers & Aliasing

- Worst case: Exponential!
- C2BP uses a pointer analysis to improve the precision of the WP(op, Q) computation.
- If the pointer analysis says that x and y cannot be aliases, only one branch of the $\vee$ is needed.

Challenges of Predicate Abstraction in C

- Pointers
- **Procedures & Procedure Calls**
- Unknown Values
- Precision-efficiency tradeoff

Challenge - Procedure & Procedure Calls

- Procedure Calls can be challenging when there are pointers.
  - Needs to update those that may have been modified by the function)
- Two Passes
  1. Generate signatures of each function in isolation.
  2. Each procedure can be abstracted given only the signatures of the abstractions of its callees.
- Modular

Challenge - Procedure & Procedure Calls

- A signature of a procedure P is: // P' is its BP(P, E)
  1. F_P, the set of formal parameters of P
  2. r, the return variable of P
  3. E_f, the set of formal parameter predicates of P'
  4. E_r, the set of return predicates of P'

Challenge - Procedure & Procedure Calls

- E_f is the subset of predicates that do not refer to any local variables of R.
- E_r contains those predicates that mention return variable but do not mention any (other) locals, as callers will not know about these locals.
- For a call of form v := P(a_1, a_2, ..), any predicate that mentions
  - v / a global variable / a (possibly transitive) dereference of an actual parameter to the call
- must be updated.

Challenges of Predicate Abstraction in C

- Pointers
- Procedures & Procedure Calls
- **Unknown Values**
- Precision-efficiency tradeoff

Challenge - Unknown Values

- Some effect in C may be hard to determine.
- So they just use "*" to represent non-deterministic, as that in
  - if (*) { assume(…) … }

Challenges of Predicate Abstraction in C

- Pointers
- Procedures & Procedure Calls
- Unknown Values
- **Precision-efficiency tradeoff**

Challenge -
Precision vs. Efficiency

- Running time of C2BP is dominated by the cost of theorem proving.
    - Worst case is exponential.
- Several optimizations to reduce the number of calling a theorem prover.

1. If a subset of formula can already imply ϕ, the whole formula implies ϕ
2. Update values of boolean variable only when necessary
3. Reduce the number of boolean variables.
4. Use syntactic heuristics.

Outline

- Main contribution
- Introduction to C2BP
- Challenges of Predicate Abstraction in C
- **Conclusion**

Conclusion - Effectiveness

- Used in the SLAM toolkit to check temporal safety properties of Windows NT device drivers.

program	lines	predicates	thm. prover calls	runtime (seconds)
floppy	6500	23	5509	98
ioctl	1250	5	500	13
openclos	544	5	132	6
srdriver	350	30	3034	93
log	236	6	98	5

Table 1: The device drivers run through C2BP.

- Discover invariants regarding array bounds checking and list-manipulating code.

program	lines	predicates	thm. prover calls	runtime (seconds)
kmp	75	4	286	7
qsort	45	2	199	5
partition	55	4	263	9
listfind	37	6	4412	172
reverse	73	7	26769	747

Table 2: The array and heap intensive programs analyzed with C2BP.

Conclusion

- Their approach may also be used to deal with other real world languages while applying predicate abstraction.

- C2BP only handles given predicates.

  - They have another tool NEWTON to generate and refine predicates automatically.

- Only for single-thread programs (at least in this paper).

Outline

- Main contribution
- Introduction to C2BP
- Challenges of Predicate Abstraction in C
- Conclusion
- **Questions?**