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
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;
    }
    newl = curr;
} else {
    // code
}

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 (*) {
                *l = nextCurr;
            }
        }
        if (*) {
            skip;
        }
        nextCurr = curr->next;
        if (prev != NULL) {
            prev->next = nextCurr;
        }
        if (curr == *l) {
            *l = nextCurr;
        }
        curr->next = newl;
    }
    newl = curr;
} else {
    // code
}
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”: $\forall P. \{P\} op \{Q\}, P \Rightarrow WP(op, Q)$

- 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 $WP(x:=e, \phi)$ where $y$ is a pointer mentioned in $\phi$
  - $\phi[x, e, y] = (\&x = \&y \land \phi[e/y]) \lor (\&x \neq \&y \land \phi)$

- 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 \( WP(\text{op, Q}) \) computation.

- If the pointer analysis says that \( x \) and \( y \) cannot be aliases, only one branch of the \( \lor \) 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 $\phi$, the whole formula implies $\phi$
  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.

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

<table>
<thead>
<tr>
<th>program</th>
<th>lines</th>
<th>predicates</th>
<th>thm. prover calls</th>
<th>runtime (seconds)</th>
</tr>
</thead>
<tbody>
<tr>
<td>floppy</td>
<td>6500</td>
<td>23</td>
<td>5509</td>
<td>98</td>
</tr>
<tr>
<td>ioctl</td>
<td>1250</td>
<td>5</td>
<td>500</td>
<td>13</td>
</tr>
<tr>
<td>openclns</td>
<td>544</td>
<td>5</td>
<td>132</td>
<td>6</td>
</tr>
<tr>
<td>srdriver</td>
<td>350</td>
<td>30</td>
<td>3034</td>
<td>93</td>
</tr>
<tr>
<td>log</td>
<td>236</td>
<td>6</td>
<td>98</td>
<td>5</td>
</tr>
</tbody>
</table>

Table 1: The device drivers run through C2BP.

<table>
<thead>
<tr>
<th>program</th>
<th>lines</th>
<th>predicates</th>
<th>thm. prover calls</th>
<th>runtime (seconds)</th>
</tr>
</thead>
<tbody>
<tr>
<td>kmp</td>
<td>75</td>
<td>4</td>
<td>286</td>
<td>7</td>
</tr>
<tr>
<td>qsort</td>
<td>45</td>
<td>2</td>
<td>199</td>
<td>5</td>
</tr>
<tr>
<td>partition</td>
<td>55</td>
<td>4</td>
<td>263</td>
<td>9</td>
</tr>
<tr>
<td>listfind</td>
<td>37</td>
<td>6</td>
<td>4412</td>
<td>172</td>
</tr>
<tr>
<td>reverse</td>
<td>73</td>
<td>7</td>
<td>26769</td>
<td>747</td>
</tr>
</tbody>
</table>

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?