Progess Report 2

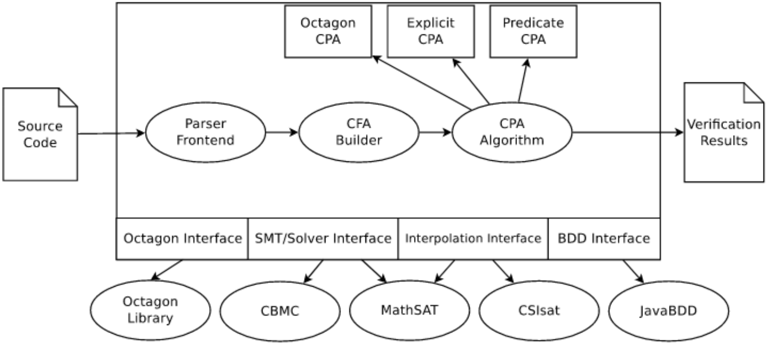
Xie Li

September 8, 2020

Overview of the Progress

- A closer look into the source code of CPACHECKER.
- Towards Practical Predicate Analysis
- Investigation of algorithm ABE (Adjust-Block Encoding).

Architecture of CPAChecker



The tool is developed in JAVA.

Parser Frontend

The function of Parser Frontend module is converting the C, java or llvm IR into data structure CFA.

- C and JAVA: use toolchain in the eclipse CDT to parse the src code into AST and then convert to nodes of CFA.
- LLVM: use their own developed package for llvm parsing.

Data Structures

- CFA: is a interface which provides methods acquiring nodes, function heads, main function and information about loop structures.
  CFA can be constructed using parsing results.
- SpecificationProperty: a class giving entry of function, properties and specifications.

```
REACHABILITY_LABEL("G ! label(ERROR)"),

REACHABILITY("G ! call(__VERIFIER_error())"),

REACHABILITY_ERROR("G ! call(reach_error())"),

VALID_FREE("G valid-free"),

VALID_DEREF("G valid-deref"),

VALID_MEMTRACK("G valid-memtrack"),

VALID_MEMCLEANUP("G valid-memcleanup"),

OVERFLOW("G ! overflow"),

DEADLOCK("G ! deadlock"),

TERMINATION("F end"),
```

```
COVERAGE_BRANCH("COVER EDGES(@DECISIONEDGE)"),

COVERAGE_CONDITION("COVER EDGES(@CONDITIONEDGE)"),

COVERAGE_STATEMENT("COVER EDGES(@BASICBLOCKENTRY)"),

COVERAGE_ERROR("COVER EDGES(@CALL(__VERIFIER_error))"),
```

Algorithm

The entry of the algorithm is in file CPAChecker.java where CPAChecker use factory to create different algorithm instances based on configurations fed to the program.

- CPA Algorithm.
- Predicate Abstraction.
- CEGAR.
- Adjustable-Block Encoding...

CPA

Algorithm 1 $CPA(\mathbb{D}, e_0)$ (taken from [8])

Input: a CPA $\mathbb{D} = (D, \rightsquigarrow, \mathsf{merge}, \mathsf{stop})$,
an initial abstract state $e_0 \in E$, where E denotes
the set of elements of the lattice of D

Output: a set of reachable abstract states

Variables: a set $\mathsf{reached}$ of elements of E,
a set $\mathsf{waitlist}$ of elements of E

$\mathsf{waitlist} := \{e_0\}$
$\mathsf{reached} := \{e_0\}$
while $\mathsf{waitlist} \neq \emptyset$ **do**
  choose e from $\mathsf{waitlist}$
  $\mathsf{waitlist} := \mathsf{waitlist} \setminus \{e\}$
  for each e' with $e \rightsquigarrow e'$ **do**
    for each $e'' \in \mathsf{reached}$ **do**
      // combine with existing abstract state
      $e_{new} := \mathsf{merge}(e', e'')$
      if $e_{new} \neq e''$ **then**
        $\mathsf{waitlist} := \left(\mathsf{waitlist} \cup \{e_{new}\}\right) \setminus \{e''\}$
        $\mathsf{reached} := \left(\mathsf{reached} \cup \{e_{new}\}\right) \setminus \{e''\}$
    if $\neg\ \mathsf{stop}(e', \mathsf{reached})$ **then**
      $\mathsf{waitlist} := \mathsf{waitlist} \cup \{e'\}$
      $\mathsf{reached} := \mathsf{reached} \cup \{e'\}$
return $\mathsf{reached}$

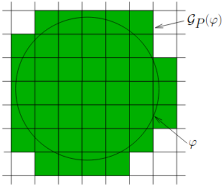
Predicate abstraction - Key operation

PREDICATE ABSTRACTION-KEY OPERATION:

- INPUT:
  - A theory T
  - A formula φ (representing, e.g., a set of concrete states)
  - A set of predicates $P = \{P_1, \ldots, P_n\}$ describing some set of properties of the system state
- OUTPUT: the most precise T-approximation of φ using P

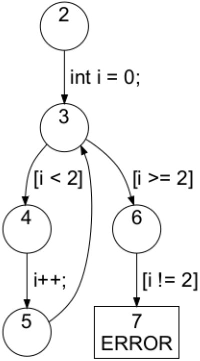
This amounts to compute either

- $\mathcal{F}_P(\varphi)$: the weakest Boolean expression over P that T-implies φ, or
- $\mathcal{G}_P(\varphi)$: the strongest Boolean expression over P T-implied by φ

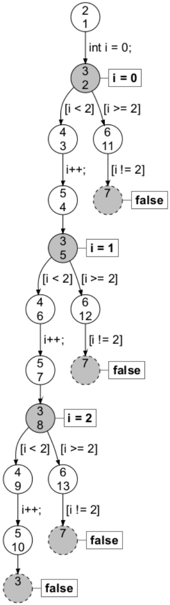


ABE Example

```
int main() {
    int i = 0;
    while (i < 2) {
        i++;
    }
    if (i != 2) {
        ERROR: return 1;
    }
}
```



ABE example



2
1
int i = 0;
3
2
i = 0
[i < 2]
[i >= 2]
4
3
6
11
i++;
[i != 2]
5
4
7
false
3
5
i = 1
[i < 2]
[i >= 2]
4
6
6
12
i++;
[i != 2]
5
7
7
false
3
8
i = 2
[i < 2]
[i >= 2]
4
9
6
13
i++;
[i != 2]
5
10
7
false
3
false