Lab: Code Verification and Z3 Theorem Prover

(Week 7)

Yulei Sui, Mohamad Barbar, Hanyuan Li, Angela He

School of Computer Science and Engineering

University of New South Wales, Australia

Assignment 2

You need to implement the following six functions in `Assignment_2.cpp` or `Assignment_2.py`:

- `SSE::reachability`
- `SSE::collectAndTranslatePath`
- `SSE::handleCall`
- `SSE::handleRet`
- `SSE::handleNonBranch`
- `SSE::handleBranch`
- The required implementation parts are indicated with TODO comments and you only need to fill up the code template if a method is partially implemented.

Python `handleIntra` implementations

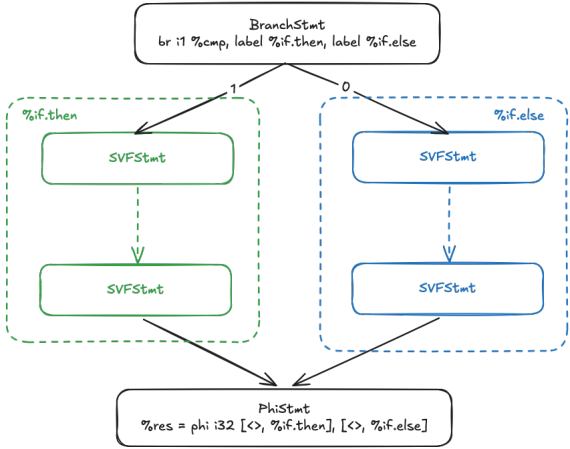
In C++, implementations for handling the intraprocedural nodes `BinaryOpStmt`, `PhiStmt` and `SelectStmt` are in the initial template, which are not in Python.

Reference implementations for these statements (for parity between C++ and Python implementations) can be found here:
`https://webcms3.cse.unsw.edu.au/COMP6131/26T2/resources/124132`

ICFG branch structure

```
int res;

if (cmp) {
    res = /* ... */;
} else {
    res = /* ... */;
}
```



Adding condition constraints

- Get the condition variable (e.g. 1 for `%if.then`)
- Check if path is feasible with existing set of constraints
- If feasible, add new constraint

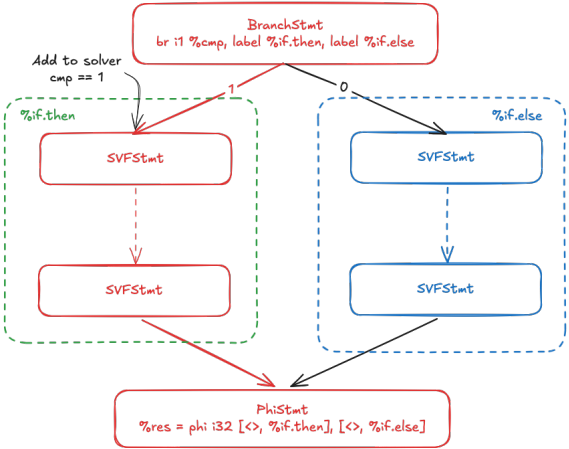
Adding condition constraints

- Get the condition variable (e.g. 1 for `%if.then`)
- Check if path is feasible with existing set of constraints
- If feasible, add new constraint

Tip

Use `solver.check()` to check if existing constraints are satisfiable, and print either the solver (C++) or use `solver.sexpr()` (Python) to output all current constraints.

Adding condition constraints



Backtracking

- You are adding branch-specific constraints (e.g. `cmp == 1`) to your solver.

Backtracking

- You are adding branch-specific constraints (e.g. `cmp == 1`) to your solver.
- When you backtrack out of the branch, those constraints no longer apply.

Backtracking

- You are adding branch-specific constraints (e.g. `cmp == 1`) to your solver.
- When you backtrack out of the branch, those constraints no longer apply.
- Make sure that when you explore other branches in your DFS, your old branch-specific constraints are no longer there.

Backtracking

- You are adding branch-specific constraints (e.g. `cmp == 1`) to your solver.
- When you backtrack out of the branch, those constraints no longer apply.
- Make sure that when you explore other branches in your DFS, your old branch-specific constraints are no longer there.

Tip

Print out your constraints to check whether the constraints you have at each point in time are correct and not excessively restrictive!

Handling Calls

At some point you will need to add a constraint like

$$<[c], rhs> \equiv <[c'], lhs>$$

Handling Calls

At some point you will need to add a constraint like

$$<[c], rhs> \equiv <[c'], lhs>$$

Remember, `getZ3Expr` is context-sensitive!

Handling Calls

At some point you will need to add a constraint like

$$<[c], rhs> \equiv <[c'], lhs>$$

Remember, `getZ3Expr` is context-sensitive!

For every callPE...

1. Grab the RHS Z3 expr (`getZ3Expr`),
   - (`getOpVarID(i)` where the call node associated with `i` matches your call node)
2. Push to your callstack (`pushCallingCtx`),
3. Grab the LHS Z3 expr (`getZ3Expr`), and
   - (`getResID`)
4. Produce your constraint (`addToSolver`).

Handling Returns

The inverse of handling calls!

Handling Returns

The inverse of handling calls!

Much simpler; are not searching/matching nodes, just `getRHSVarID()`/`getLHSVarID()` on your `RetPE`.

Handling Returns

The inverse of handling calls!

Much simpler; are not searching/matching nodes, just `getRHSVarID()`/`getLHSVarID()` on your `RetPE`.

Be aware of functions which return nothing!