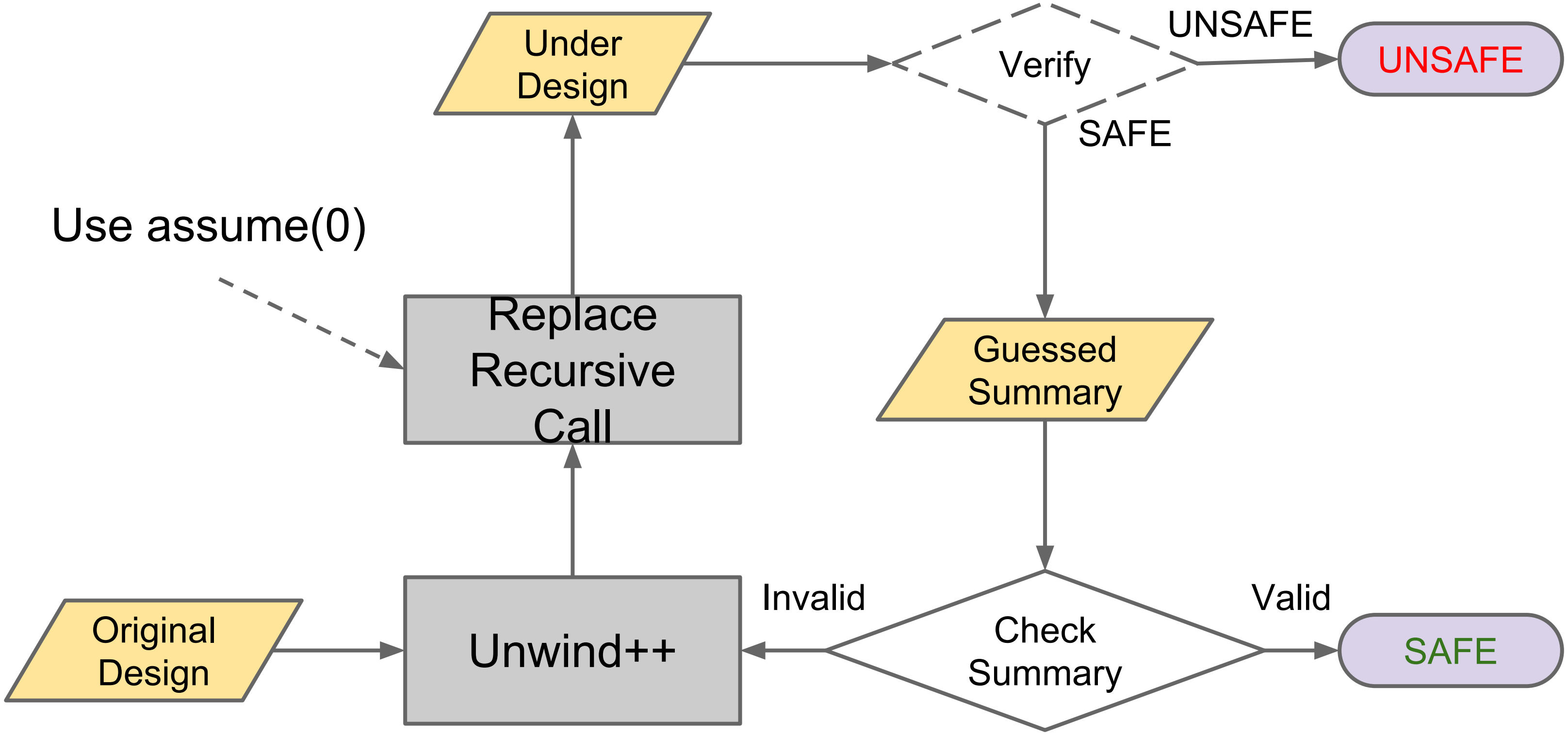
Progress Notes

Jan. 16 ~ 23, 2014

Original Flow

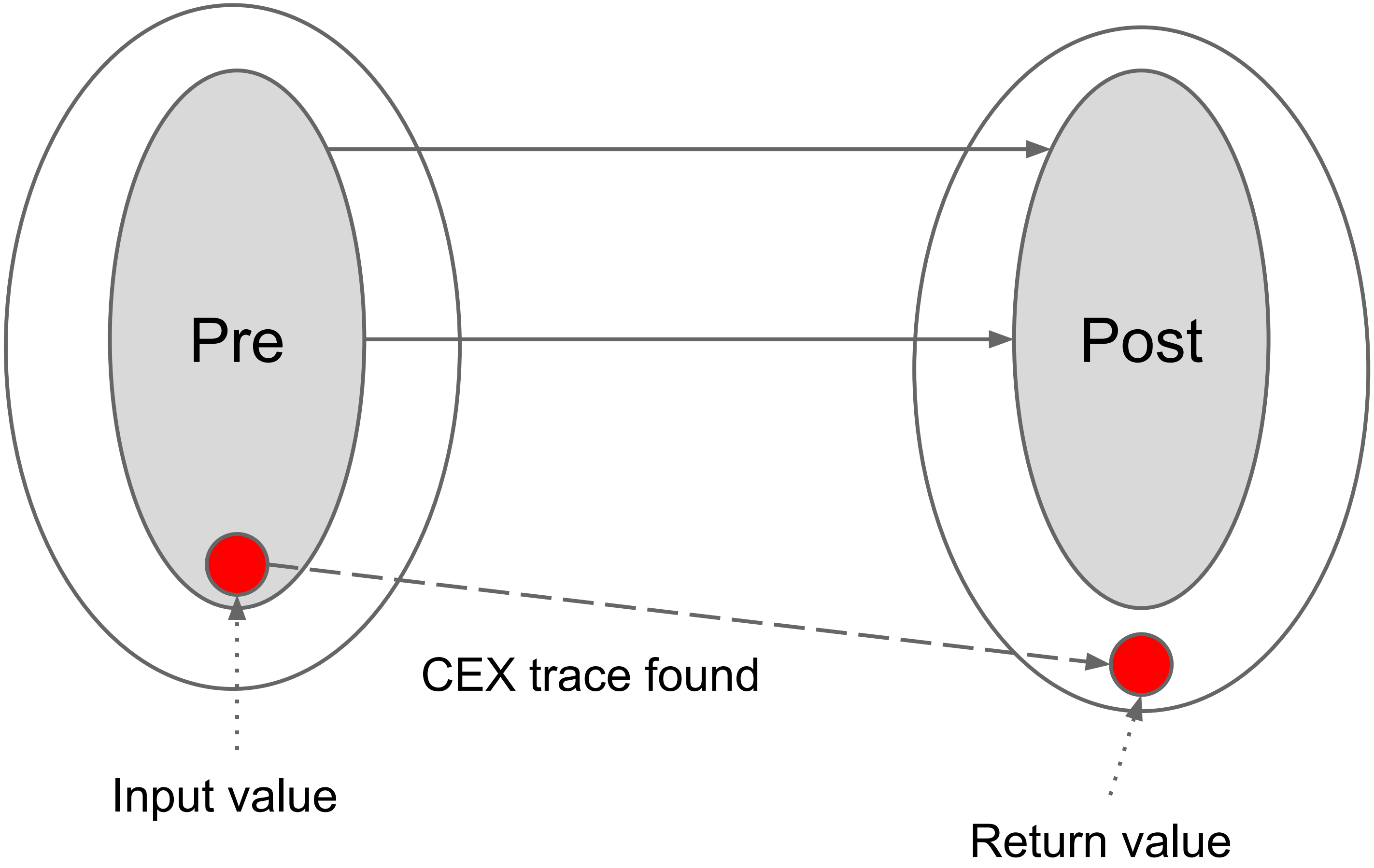


Problems

- Two interesting cases
  - recHanoi03_true.c
  - Fibonacci01_true.c

- Failed at Check Summary
  - Summary is discarded in our current flow
  - Need methods to reuse the summary
    - Add more information from original design

Summary Failed at check



First Case

recHanoi03_true

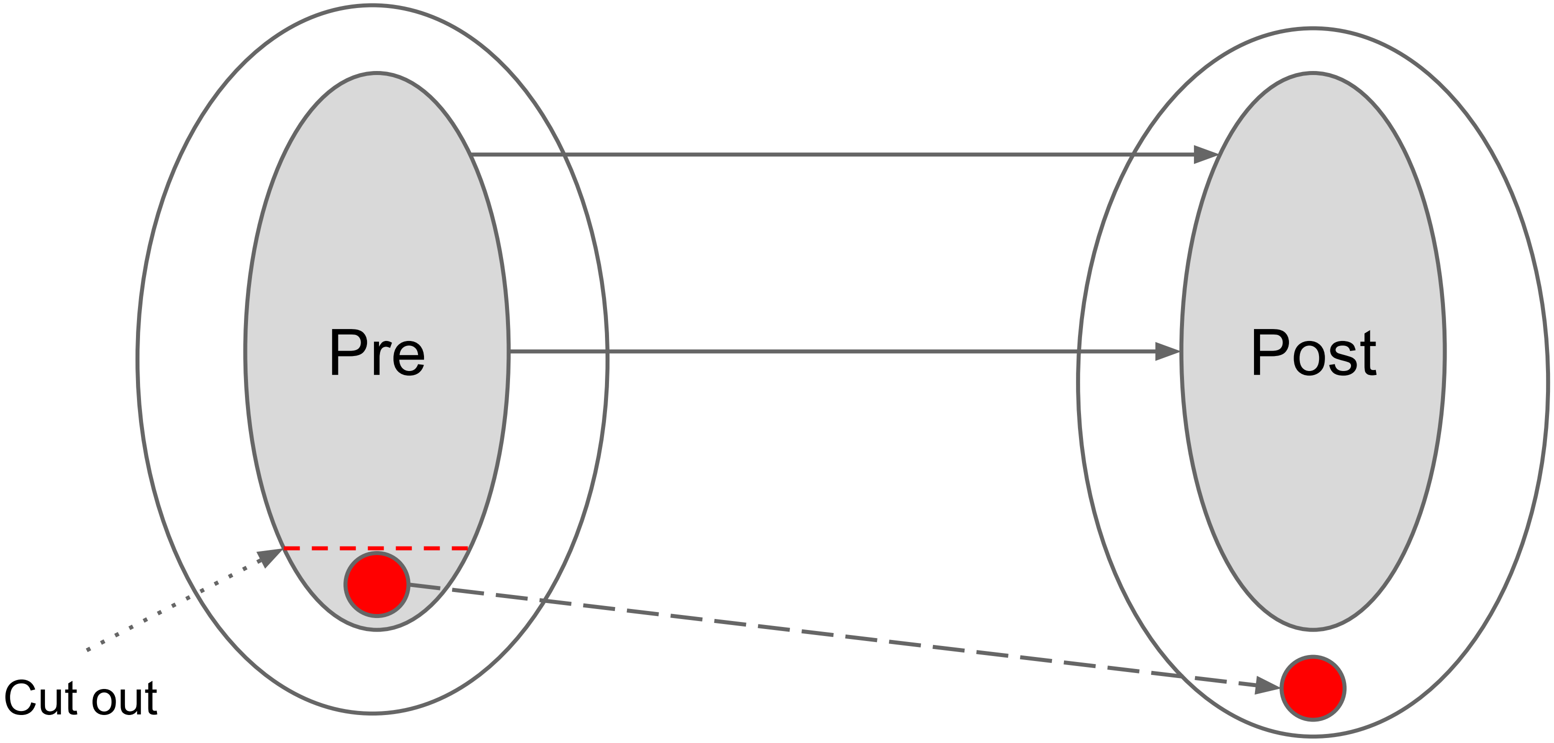
```
int hanoi(int n) {
    if (n == 1) { return 1;}
    return 2 * (hanoi(n-1)) + 1;
}
int main() {
    int n = _nondet_int();
    if (n < 1 || n > 31) { return 0;} // Contraint on
input

    int result = hanoi(n);
    assert(result >= n);

    return 0;
}
```

Guess Summary and Check

- Guessed Summary
  - Pre-CON = true
  - Post-CON = (r >= n)
  - Summary = (true) => (r >= n)

- Check Summary
  - Failed
  - CEX: n=0, r=-1;
  - CEX is caused by impossible input value.

Exclude input value of CEX



Refine Pre-condition

- Add assertions at main()
  - Add assert(! <input value>) before all rec() calls

- Verify main() by verifier
  - SAFE

    => Such input value is not used.

    => Get new summary from SAFE ARG
    => New summary as another guess
  - UNSAFE

    => Such input value is possible
    => Need more discussion

Case 1 ~ recHanoi03_true.c

- Manually tested
- 1st Guess
  - Summary: (true) => (r >= n)
  - => Check Summary Failed
  - CEX: n=0; r=-1;
- 2nd Guess
  - Manually add assert(!(n=-1))
  - Summary: (1<=n) => (r >= n)
  - => Check Summary Passed

Second Case

Fibonacci01_true

```
int fibonacci(int n) {
    if (n < 1) {
        return 0;
    } else if (n == 1) {
        return 1;
    } else {
        return fibonacci(n-1) + fibonacci(n-2);
    }
}
```

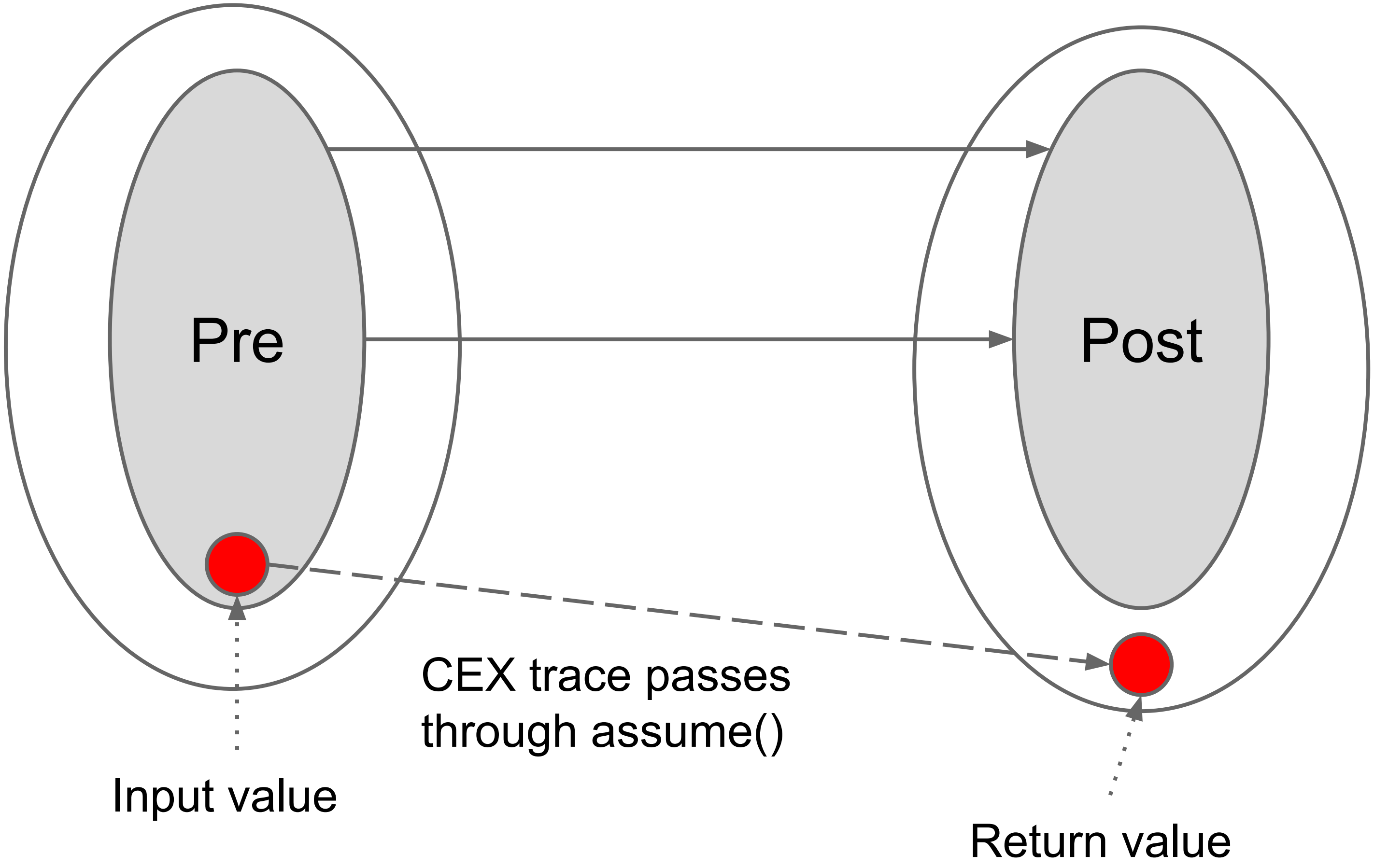
```
Assertion: (r >= n - 1)
```

Guess Summary and Check

- Guessed Summary
  - Pre-CON = true
  - Post-CON = (n - r <= 1)
  - Summary = (true) => (n - r <= 1)

- Check Summary
  - Failed
  - CEX: n=2, r=-1;
  - CEX is caused by fake transition.

Summary Failed at check



Unwind function body

- Observe found CEX trace
    - CEX trace must pass through assumed summary.

- Use unwinded version of function body
    - Provide more accurate transition function
    - Avoid fake transitions caused by assumed summary.
    - Use the same unwinded version used in main()
        - No changes on original flow

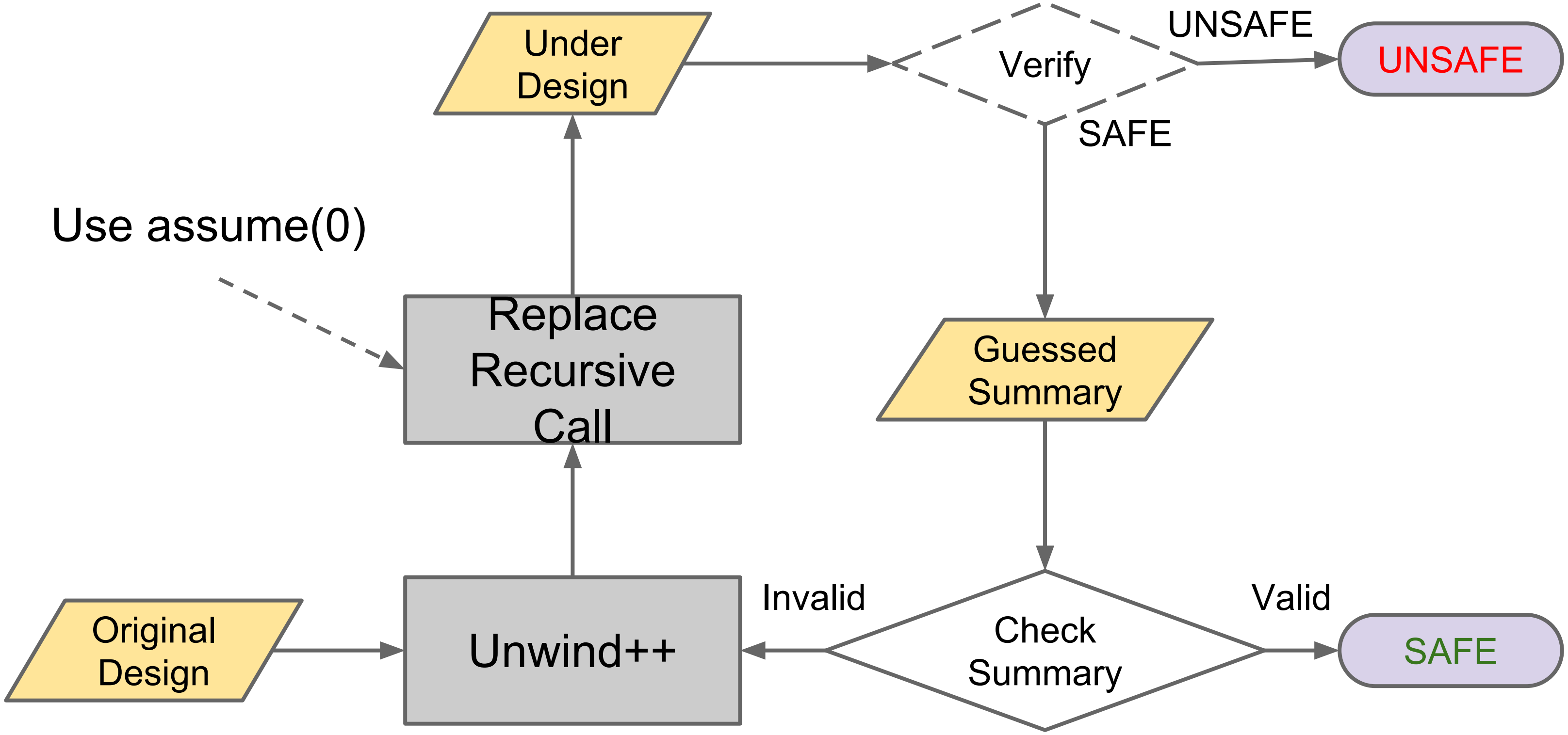
Case 2 ~ Fibonacci01_true.c

- Assertion: (ret >= n - 1)

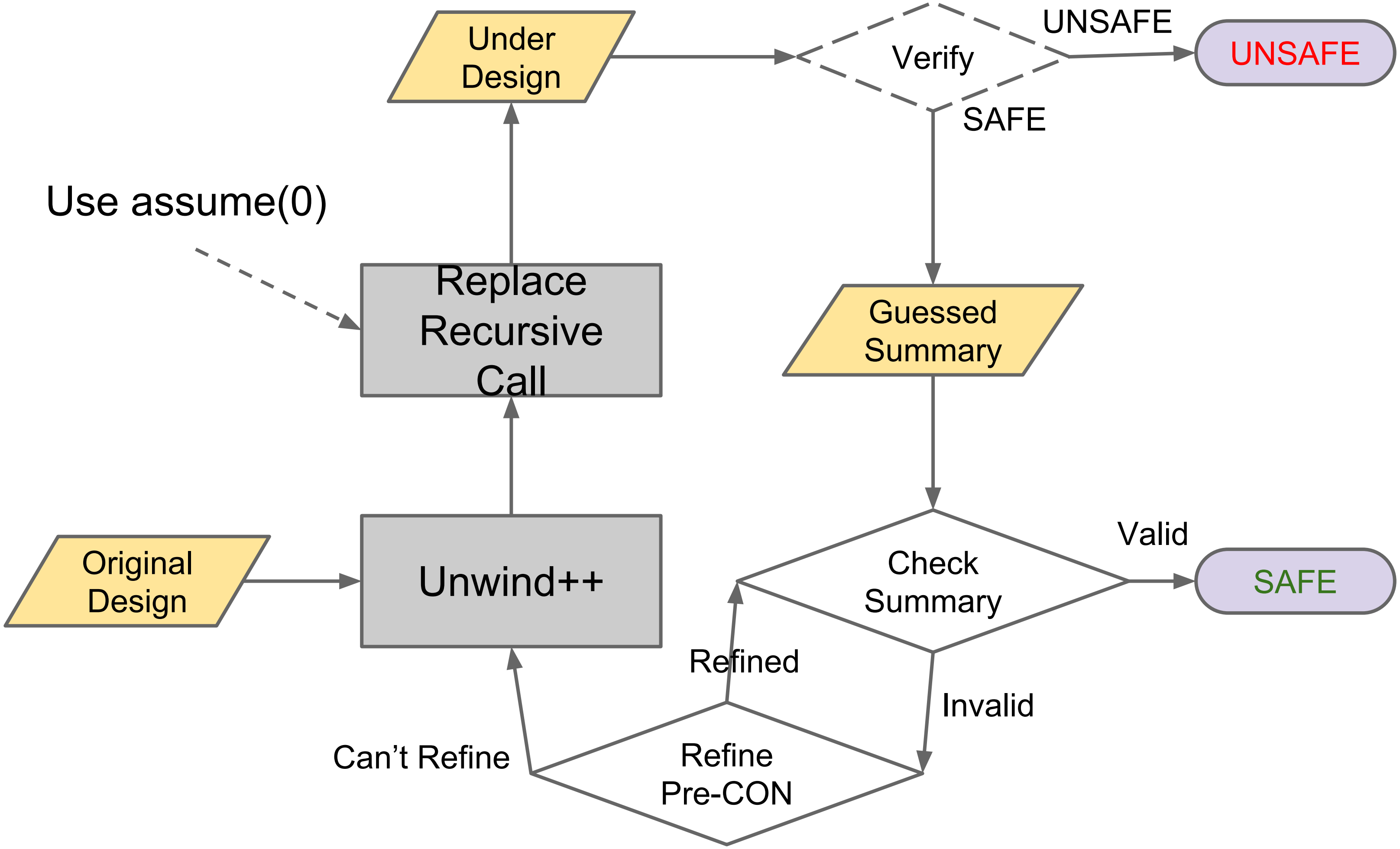
- Guess: (n - ret <= 1)
  - Pass when the function body is unwinded 5 times

Add to Flow

Original Flow



Modified Flow



Progress Notes

Jan. 24 ~ 28, 2014

Question

- Assertion for checking pre-condition
  - When checking summary
  - Need to check that input value of recursive call are always in pre-condition?

```
int hanoi(int n){
    if(n == 1){ ret = 1; goto RET;}

    n_1 = n - 1;

    tmp_2 = hanoi(n_1);

    ret = 2 * tmp_2 + 1; goto RET;
RET:
    return ret;
}
```

Example ~ No Pre-Condition

```
int hanoi(int n){
    // assume(true);

    if(n == 1){ ret = 1; goto RET;}

    n_1 = n - 1;

    // tmp_2 = hanoi(n_1);
    // assert(true);
    tmp_2 = nondet_int();
    assume(!(n_1 >= 1) || tmp_2 >= n_1);

    ret = 2 * tmp_2 + 1; goto RET;
RET:
    assert(!(n >= 1) || ret >= n);
    return ret;
}
```

Example ~ With Pre-Condition

```
int hanoi(int n){
    assume(n >= 1);

    if(n == 1){ ret = 1; goto RET;}

    n_1 = n - 1;

    // tmp_2 = hanoi(n_1);
    assert(n_1 >= 1);
    tmp_2 = nondet_int();
    assume(!(n_1 >= 1) || tmp_2 >= n_1);

    ret = 2 * tmp_2 + 1; goto RET;
RET:
    assert(!(n >= 1) || ret >= n);
    return ret;
}
```

My Answer

- No need to check pre-condition
  - Summary = (Pre-CON => Post-CON)
  - assume(Summary) for replace function call
    - For input value not in Pre-CON,
      return value can be any value
  - assert(Summary) at return location
    - For input value in Pre-CON,
      Check return value is in Post-CON
    - For input value not in Pre-CON,
      return value is not checked
    - We don’t care return values of these input values