Karlsruhe Institute of Technology

Formal Systems II: Applications

Bounded Model Checking of C Programs and LLBMC

Carsten Sinz | SS 2017

Institute for Theoretical Computer Science
Research Group “Verification meets Algorithm Engineering”

What is Bounded Model Checking?

- Background
  - Restrict programs, such that property checking becomes decidable
  - Fully automatically check these programs for typical programming errors
  - Focus on „real programs“, not „algorithms“ --> machine integers
- Example:

```
int abs(int x)
{
    int ret;
    if (x >= 0) {
        ret = x;
    } else {
        ret = -x;
    }
    __llbmc_assert(ret >= 0);
    return ret;
}
```

Software Bounded Model Checking (SBMC)

- Decision procedure for reachability properties of programs
- Decision procedure
  - Restrict programs
    - No infinite program runs
  - Cut off analysis of program paths after some limit is reached
    - Typical: after a fixed number of loop iterations, recursive calls
  - Model machine arithmetic precisely
    - Bounded integer arithmetic is closely related to SAT
  - Result: Everything is finite!
- Properties
  - Reachability (i.e. can a statement be reached in a program run)
  - Not termination etc.
- Programs
  - In a real programming language, mainly C/C++

SBMC by Example

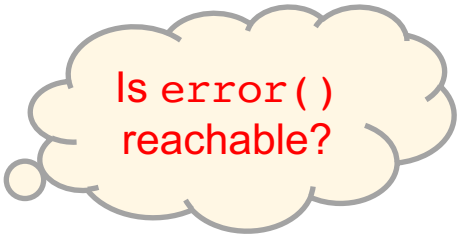
- Runtime errors

```
const int MAX_DIFF = 100;      // used to be 80
extern int brake_intensity[6];


int auto_brake_intensity(int speed_diff)
{
    int diff_in_mps, b_idx;


    if (speed_diff > MAX_DIFF || speed_diff < 0) {
        return 0;
    }

    diff_in_mps = speed_diff * 1000 / 3600;
    b_idx = diff_in_mps >> 2;


    return brake_intensity[b_idx];
}
```

SBMC by Example

- Arithmetic

```
unsigned int
square_check(unsigned int x)
{
    unsigned int y = x * x;
    if (y == 33) { error(); }
    return y;
}
```



$\exists x \in \mathbb{Z}/2^{32}\mathbb{Z}\colon\ x^2 = 33$?

4 solutions, e.g. 633.169.809

SBMC by Example

- Functional property checking with assert/assume

```
int npo2(int x)
{
    unsigned int i;
    x--;
    for(i=1; i < sizeof(int)*8; i *= 2)
        x = x | (x >> i);
    return x+1;
}
```

```
void __llbmc_main(int x)
{
    // restrict input
    __llbmc_assume(x <= 16384 && x > 8192);

    int n = npo2(x);

    // expected output
    __llbmc_assert(n == 16384);
}
```

SBMC by Example

- Equivalence Checking

```
uint32_t reference_popcount(uint32_t x)
{
    int i, s = 0;
    for (i = 0; i < 32; ++i)
        if (x & (1 << i))
            s++;
    return s;
}

uint32_t optimized_popcount(uint32_t x)
{
    x = x - ((x >> 1 ) & 0x55555555);
    x = (x & 0x33333333) + ((x >> 2) & 0x33333333);
    x = (x + (x >> 4)) & 0x0F0F0F0F;
    x += (x >> 8);
    x += (x >> 16);
    return x & 0x0000003F;
}

void __llbmc_main(uint32_t x)
{
    uint32_t opt = optimized_popcount(x);
    uint32_t ref = reference_popcount(x);
    __llbmc_assert(opt == ref);
}
```