SMT-based Bounded Model Checking for Multi-threaded Software in Embedded Systems

Supervisor: Dr. Bernd Fischer

Lucas Cordeiro
lcc08r@ecs.soton.ac.uk
Embedded systems are ubiquitous but their verification becomes more difficult.

- functionality demanded increased significantly
  - peer reviewing and testing
- multi-core processors with scalable shared memory
  - but most verification tools focus on message passing

```c
void *threadA(void *arg) {
    lock(&mutex);
    x++;
    if (x == 1) lock(&lock); (CS1)
    unlock(&mutex);
    lock(&mutex); (CS3)
    x--;
    if (x == 0) unlock(&lock);
    unlock(&mutex);
}

void *threadB(void *arg) {
    lock(&mutex);
    y++;
    if (y == 1) lock(&lock); (CS2)
    unlock(&mutex);
    lock(&mutex);
    y--;
    if (y == 0) unlock(&lock);
    unlock(&mutex);
}
```
Scalability and Precision in Bounded Model Checking for ANSI-C

• state space explosion problem
  – exploit proof of unsatisfiability
  – integrate POR with symbolic algorithms
    → visible instruction and read-write analysis
• precision of arithmetic and bit-level operations
  – use decision procedures of QF formulae with a more accurate model of the ANSI-C semantics (SMT)
    → combine different background theories and solvers

Can an algorithmic method reason accurately about multi-threaded software in embedded systems by controlling the verification complexity?
SMT-based Verification of Multi-threaded Software

Lazy exploration of interleavings

#define N 10
int a [N] , i , j =1, x=2;
void *Tx(void *arg) {
    if (x>2) {
        a[i] = *((int *)arg);    //X0
        assert(i>=0 && i<N);     //X1
    }
}
void *Ty(void *arg) {
    if (x>3) {
        a[j]=*((int *)arg);     //Y0
    } else
        x=3;                         //Y1
}
int main(void) {
    int arg1=10, arg2=20; i=nondet_uint();
    //create Tx with arg1 //create Ty with arg2
}
SMT-based Verification of Multi-threaded Software

Scheduling Recording

#define N 10
int a[N], i, j = 1, x = 2;
void *Tx(void *arg) {
    if (x > 2) {
        a[i] = *((int *)arg);    //X0
        assert(i >= 0 && i < N);  //X1
    }
}
void *Ty(void *arg) {
    if (x > 3)  
        a[j] = *((int *)arg);   //Y0
    else
        x = 3;                       //Y1
}
int main(void) {
    int arg1 = 10, arg2 = 20;
    i = nondet_uint();
    //create Tx with arg1
    //create Ty with arg2
}
SMT-based Verification of Multi-threaded Software

Under-approximation & Widening

#define N 10
int a[N], i, j = 1, x = 2;
void *Tx(void *arg) {
    if (x > 2) {
        a[i] = *((int *)arg);  //X0
        assert(i >= 0 && i < N);
        //X1
    }
    void *Ty(void *arg) {
        if (x > 3) {
            a[j] = *((int *)arg);  //Y0
            x = 3;  //Y1
        } else {
            x = 3;
        }
    }
}
int main(void) {
    int arg1 = 10, arg2 = 20;
    i = nondet_uint();
    //create Tx with arg1
    //create Ty with arg2
    return 0;
}
Experimental Evaluation

• described and evaluated SMT-based BMC in large embedded software
  – SMT-based BMC is more efficient than SAT-based BMC (but not uniformly)
  – introduced continuous verification for large systems
• evaluated the UW, schedule recording, and lazy approaches
  – add concurrency constraints lazily
    → extremely fast for satisfiable instances
  – memory overhead and slowdowns to extract the unsatisfiable cores
Results

• built and evaluated first SMT-based BMC for ANSI-C
• **UW, lazy** and **schedule recording** algorithms
• introduced **continuous verification** approach
• [users.ecs.soton.ac.uk/lcc08r/esbmc/](users.ecs.soton.ac.uk/lcc08r/esbmc/)

Future Work

• partial order reduction (static and dynamic)
• data races detection (compatibility with compiler)
• Craig interpolation to generate threads scheduling