Using SPIN to Model Check Concurrent Algorithms, using a translation from C to Promela

Ke Jiang
Bengt Jonsson
Uppsala University
1. Introduction

1.1 Background

- To achieve performance, complex concurrent algorithms have been developed, but are hard to get correct, e.g. concurrent queue algorithms
- Model checking
- SPIN model checker and Promela

1.2 Motivation

- Use SPIN to automatically analyze concurrent algorithms written in (a significant subset of) C
- Automatically abstract Promela models from input C codes

1.3 Related works

Modex and our previous paper [Bengt Jonsson, MCC 08]
2. A motivating example

- Concurrent queue algorithms
- Translation from pseudocode of Michael and Scott's blocking queue algorithm (also in our previous work)
- Such algorithms will likely cause bottlenecks in concurrent programs
```c
struct node_t {
    int value;
    struct node_t *next;
};

struct queue_t {
    struct node_t *Head;
    struct node_t *Tail;
    int H_lock;
    int T_lock;
};

void enqueue(struct queue_t *Q, int val) {
    struct node_t *node = malloc(sizeof(struct node_t));
    node->value = val;
    node->next = 0;
    lock(&Q->T_lock);
    Q->Tail->next = node;
    Q->Tail = node;
    unlock(&Q->T_lock);
}
```

- `node_t` and `queue_t` are used to create linked lists
- The two locks in `queue_t` allow concurrent access
- Other algorithms might use **CAS** instead of locks
3. Promela

Promela is a modeling language designed for state space exploration of finite-state models.

- Promela has C-like syntax
- Processes
- Basic data types
- Control flows
- Communications
- No dynamically allocated data structures
- No functions and function calls
4. Implementation

CIL (C Intermediate Language): the carrier of our translator

- Handles the parsing and semantic analysis of C programs
- Compiles valid C programs into core constructs (e.g. the loops)
- Can be guided to perform user defined transformation (e.g. the translator we developed)
- Provides C syntax trees for our translator
4. Implementation

The whole translation flow
5. Syntax-directed translation

5.1 Data Structures and Statements

Translate in a straight-forward fashion:

✓ Simple statements and control structures
✓ Integers and arrays
✗ Pointers and structures
typedef record {
    int min;
    int max;
};

... struct record r;
for (x=0; x<5; x++)
    for (y=0; y<4; y++)
        if (a[y] > a[y+1]) {
            temp = a[y+1];
            a[y+1] = a[y];
            a[y] = temp;
        }
    r.min = a[0];
    r.max = a[4];
...
5. Syntax-directed translation

5.2 Dynamically Allocated Memory

- Dynamically allocated memory simulated by dedicated array
- Integers and structures
- One array per data type
- All memory operations must be type-respecting
- Pointer arithmetic is not supported
5. Syntax-directed translation

5.2 Dynamically Allocated Memory

- Dynamically allocated memory simulated by dedicated array
- Integers and structures
- One array per data type
- All memory operations must be type-respecting

- Pointer arithmetic is not supported

A example of a “static” list in Promela
5. Syntax-directed translation

Translating Pointer operations

- Pointer assignments will be translated into integer assignments
- Assignments to the original variable by pointers will be translated to assignments corresponding positions of memory arrays

```
struct person{
    int age;
};

struct person *ptr;
struct person *temp;

ptr = temp;
...
    temp->age = 24;
...
```

```
typedef person {
    int age;
}

person person_mem[9];
int person_valid[9];

int ptr;
int temp;

ptr = temp;
...
    person_mem[temp].age = 24;
...
5. Syntax-directed translation

- Memory allocations using `malloc` are simulated
- Memory releases using `free` are analogous

```c
struct person{
    int age;
};
...
struct person *ptr;
ptr = malloc(sizeof(struct person));
...
free(ptr);
...
typedef person {
    int age;
}
person person_mem[9];
it person_valid[9];
...
int ptr;
$$$
//person_ct gets first available position of person_valid
ptr = person_ct;
...```
```c
D_step {
    person_valid[ptr] = 0;
    person_mem[ptr].age = 0
};
...```
5.3 Translating functions

• No function concept in Promela! Need to be emulated
• Use process to simulate functions

Function definitions: parameters and function body

Function calls: into 3 steps. Exceptions: lock and unlock

Return statements: emulate the two properties of return.

Recursive functions: are automatically handled. Recursion depth need to be noticed
int test(int a, int b) {
  if (a >= b) return a;
  else return b;
}

... int x = 2;
int y = 3;
test(x, y);
...

proctype test(chan in_test; int a; int b) {
  if
    :: (a >= b) ->
      in_test ! a;
      goto end
    :: else ->
      in_test ! b;
      goto end
  fi;
  end:
  printf ("End of test")
}

... chan ret_test = [0] of { int };
int x; int y; int tmp;
x = 2;
y = 3;
run test(ret_test, x, y);
ret_test ? tmp;
...
6. Experiments

- We denote test harnesses using a sequence of $E$ (enqueue) and $D$ (dequeue) in threads separated by $I$, e.g. $EE|DD$.
- Two verification modes: Exhaustive and Bitstate
- The following experiments were performed using Exhaustive mode

<table>
<thead>
<tr>
<th>Test harness</th>
<th>States</th>
<th>Mem.</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>E</td>
<td>D</td>
<td>851</td>
<td>5.044</td>
</tr>
<tr>
<td>EE</td>
<td>DD</td>
<td>14,467</td>
<td>11.587</td>
</tr>
<tr>
<td>EE</td>
<td>DDD</td>
<td>29,506</td>
<td>19.009</td>
</tr>
<tr>
<td>EEE</td>
<td>DDD</td>
<td>138,751</td>
<td>74.575</td>
</tr>
<tr>
<td>EEEE</td>
<td>DD</td>
<td>128,611</td>
<td>69.399</td>
</tr>
<tr>
<td>EEEE</td>
<td>DDDDD</td>
<td>1,101,416</td>
<td>583.583</td>
</tr>
<tr>
<td>EEEEEE</td>
<td>DDDDD</td>
<td>3,181,607</td>
<td>1727.196</td>
</tr>
<tr>
<td>EEEEE</td>
<td>DDDDD</td>
<td>7,894,946</td>
<td>4403.891</td>
</tr>
</tbody>
</table>
6. Experiments

- We denote test harnesses using a sequence of $E$ (enqueue) and $D$ (dequeue) in threads separated by $I$, e.g. $EE|DD$.
- Two verification modes: Exhaustive and Bitstate
- The following experiments were performed using Exhaustive mode

<table>
<thead>
<tr>
<th>Test harness</th>
<th>States</th>
<th>Mem.</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>$E</td>
<td>D$</td>
<td>851</td>
<td>5.044</td>
</tr>
<tr>
<td>$EE</td>
<td>DD$</td>
<td>14,467</td>
<td>11.587</td>
</tr>
<tr>
<td>$EE</td>
<td>DDD$</td>
<td>29,506</td>
<td>19.009</td>
</tr>
<tr>
<td>$EEE</td>
<td>DDD$</td>
<td>138,751</td>
<td>74.575</td>
</tr>
<tr>
<td>$EEEE</td>
<td>DD$</td>
<td>128,611</td>
<td>69.399</td>
</tr>
<tr>
<td>$EEEE</td>
<td>DDDD$</td>
<td>1,101,416</td>
<td>583.583</td>
</tr>
<tr>
<td>$EEEEE</td>
<td>DDDD$</td>
<td>3,181,607</td>
<td>1727.196</td>
</tr>
<tr>
<td>$EEEEE</td>
<td>DDDDDD$</td>
<td>7,894,946</td>
<td>4403.891</td>
</tr>
</tbody>
</table>
7. Conclusion

✔ Automatic translation from C to Promela makes model checking C programs using SPIN possible.

✔ Innovative solutions for “untranslatable” structures in previous works, e.g. pointers and functions.

✗ Still does not have the same capabilities comparing to our manual work, e.g. no impact of weak memory models, not as efficient.
8. Future works

- Add the impact of weak memory models
- Optimize the current translation
  - Better output layout
  - Support more complex C structures, e.g. multi-dimensional arrays, pointer to pointers
  - Divide necessary statements into atomic parts, e.g. x++
- Garbage collection