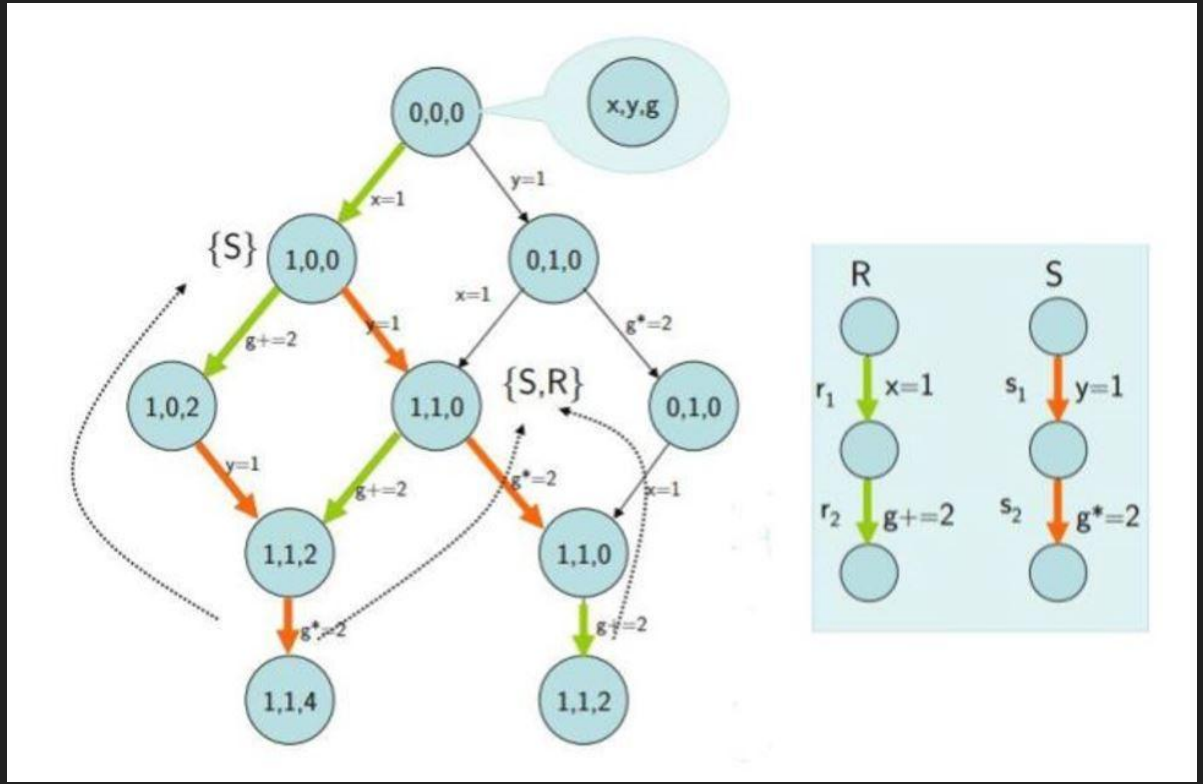
LLVM Implementation of DPOR

Xianpei Meng

Quick Review



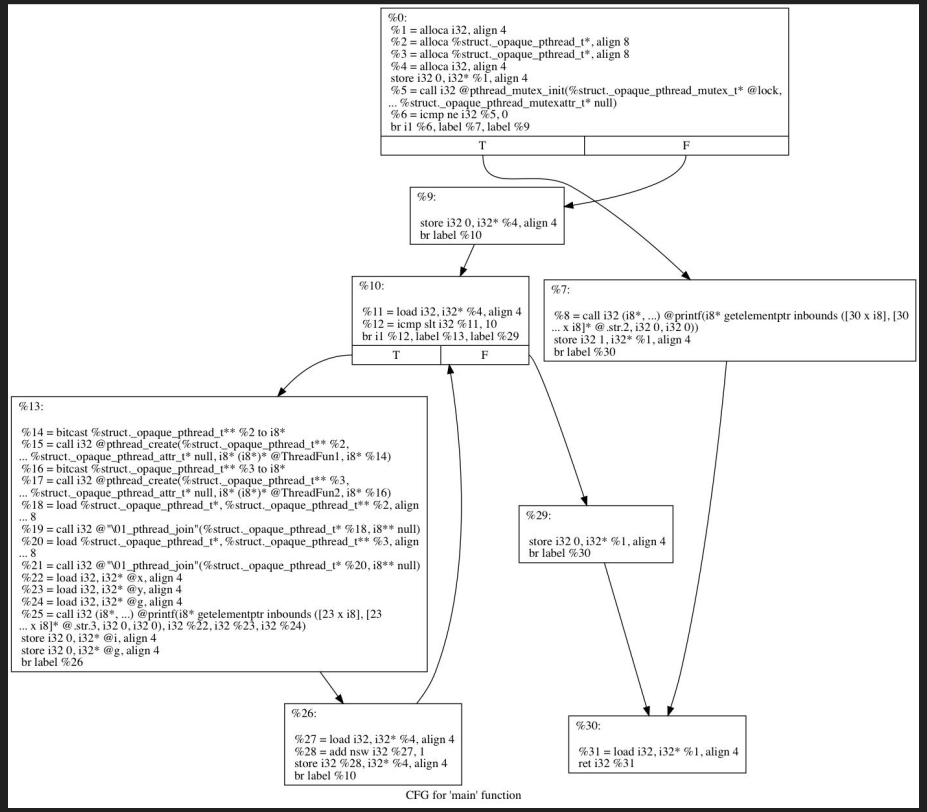
Quick Review

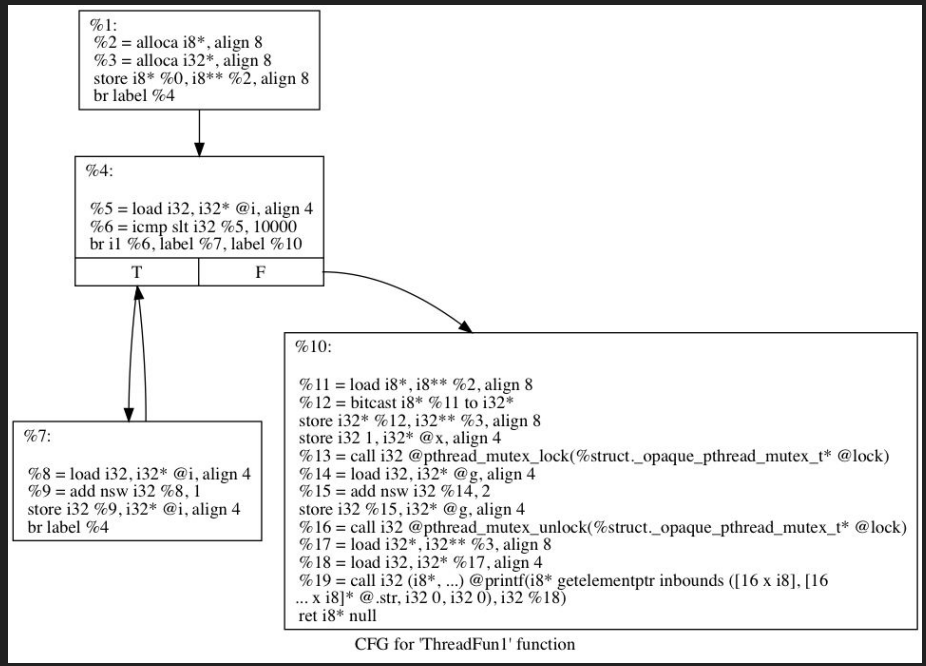
```
0   Initially: Explore(∅);

1   Explore(S) {
2       let s = last(S);
3       for all processes p {
4           if ∃i = max({i ∈ dom(S) | S_i is dependent and may be co-enabled with next(s,p) and i ↛_S p}) {
5               let E = {q ∈ enabled(pre(S,i)) | q = p or ∃j ∈ dom(S) : j > i and q = proc(S_j) and j →_S p};
6               if (E ≠ ∅) then add any q ∈ E to backtrack(pre(S,i));
7                   else add all q ∈ enabled(pre(S,i)) to backtrack(pre(S,i));
8           }
9       }
10      if (∃p ∈ enabled(s)) {
11          backtrack(s) := {p};
12          let done = ∅;
13          while (∃ p ∈ (backtrack(s) \ done)) {
14              add p to done;
15              Explore(S.next(s,p));
16          }
17      }
18  }
```

My implementation

- test1.c -a simple testbench contains 2 threads
- DPOR.cpp -llvm pass that inspects and inserts instructions
- rtlib.cpp -runtime library. Linked with test1.bc and function call inserted by LLVM pass.
- dpor.cpp -the program to control everything by using system call to run several shell script.
- record.txt -the txt file records all the execution history, served as dom(S)
- nextbt.txt -the txt file holds important variables for making decision. accessed by DPOR.cpp and dpor.cpp

My implementation -- CFG of testbench





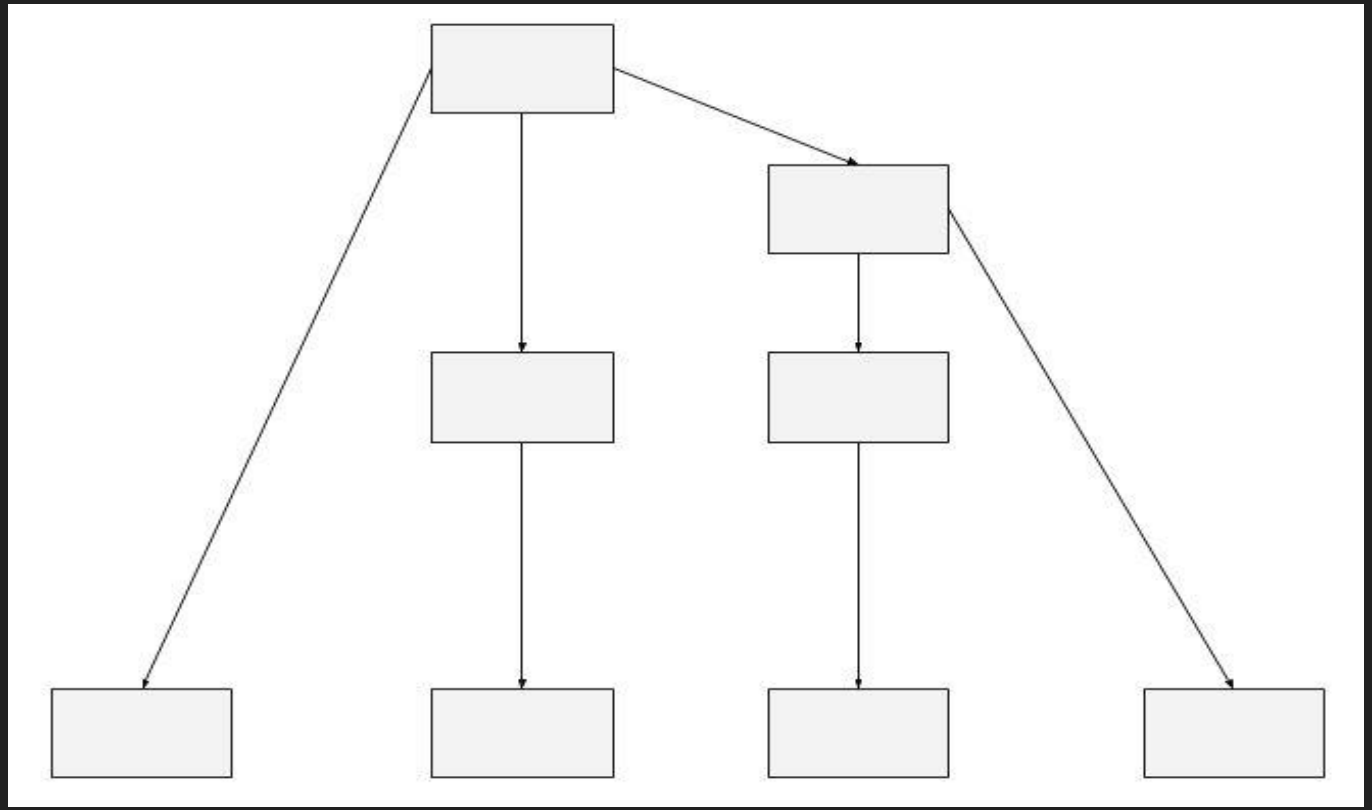
My implementation -- LLVM pass

- Using LLVM::IRbuilder to insert function calls from rtlib.cpp to test1.bc. Link both of them to generate a new executable.
- rtlib.cpp has 2 functions: first writes information to record.txt and second sleep for a certain time.(0.x sec-0.0x sec)
- Read from both record.txt and nextbt.txt to decide where to make insertion

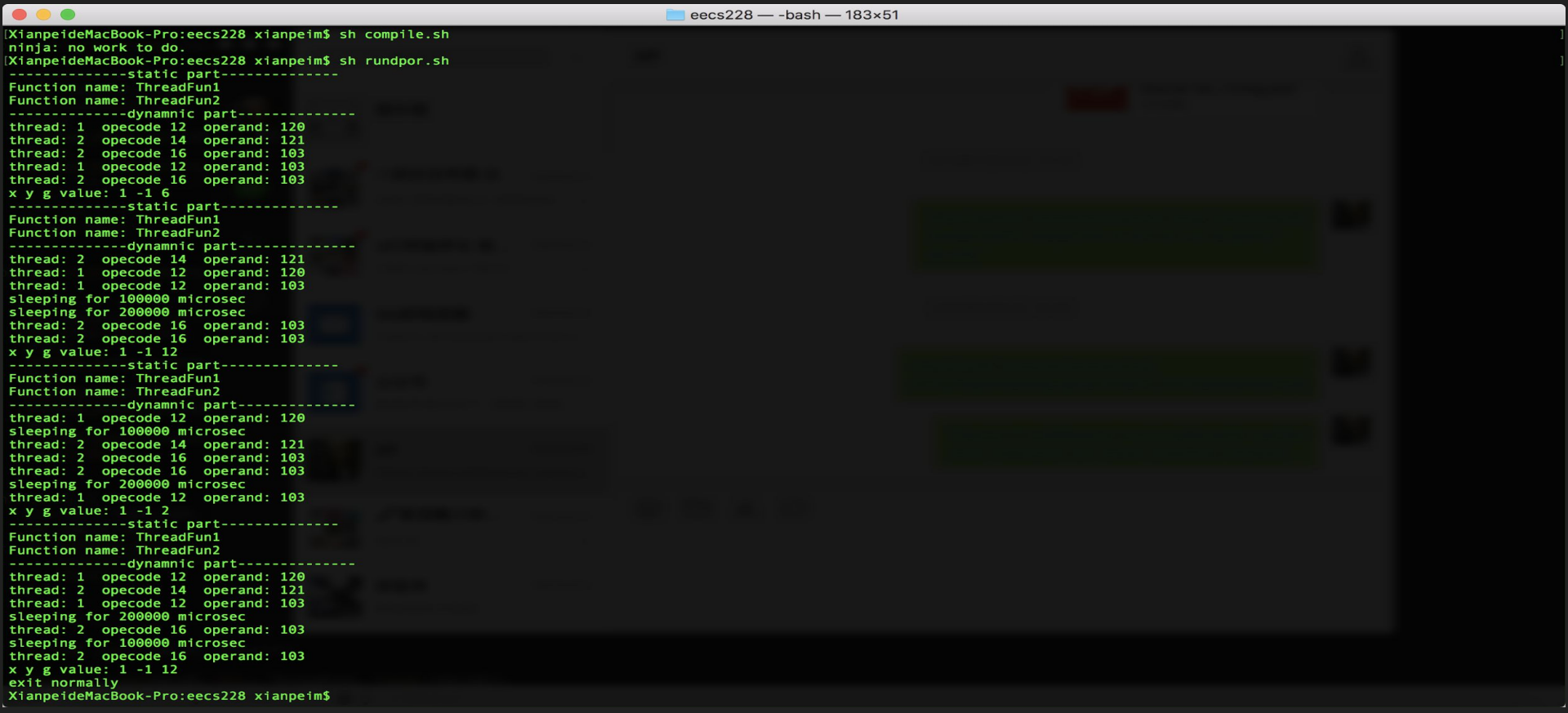
My implementation -- inserted instruction

```
81
82  ; <label>:12:                                     ; preds = %6
83    %13 = load i32, i32* @y, align 4
84    %14 = sub nsw i32 %13, 1
85    call void @record_op(i32 2, i32 14, i32 121)
86    store i32 %14, i32* @y, align 4
87    call void @delay(i32 1)
88    %15 = call i32 @pthread_mutex_lock(%struct._opaque_pthread_mutex_t* @lock)
89    %16 = load i32, i32* @g, align 4
90    %17 = mul nsw i32 %16, 2
91    call void @record_op(i32 2, i32 16, i32 103)
92    store i32 %17, i32* @g, align 4
93    %18 = call i32 @pthread_mutex_unlock(%struct._opaque_pthread_mutex_t* @lock)
94    %19 = call i32 @pthread_mutex_lock(%struct._opaque_pthread_mutex_t* @lock)
95    %20 = load i32, i32* @g, align 4
96    %21 = mul nsw i32 %20, 3
97    call void @record_op(i32 2, i32 16, i32 103)
98    store i32 %21, i32* @g, align 4
99    %22 = call i32 @pthread_mutex_unlock(%struct._opaque_pthread_mutex_t* @lock)
100   ret i8* null
101 }
```

My implementation -- LLVM pass



Example- 2 threads including 3 different operations on same shared variable



Example- 2 threads including 4 different operations on same shared variable

eecs

```
XianpeideMacBook-Pro:eecs228 xianpeim$ sh rundpor.sh
--------------static part--------------
Function name: ThreadFun1
Function name: ThreadFun2
--------------dynamnic part--------------
thread: 1  opecode 12  operand: 120
thread: 2  opecode 14  operand: 121
thread: 2  opecode 16  operand: 103
thread: 1  opecode 12  operand: 103
thread: 2  opecode 16  operand: 103
thread: 1  opecode 12  operand: 103
x y g value: 1 -1 10
--------------static part--------------
Function name: ThreadFun1
Function name: ThreadFun2
--------------dynamnic part--------------
thread: 2  opecode 14  operand: 121
thread: 1  opecode 12  operand: 120
sleeping for 100000 microsec
thread: 1  opecode 12  operand: 103
sleeping for 200000 microsec
thread: 2  opecode 16  operand: 103
thread: 2  opecode 16  operand: 103
thread: 1  opecode 12  operand: 103
x y g value: 1 -1 16
--------------static part--------------
Function name: ThreadFun1
Function name: ThreadFun2
--------------dynamnic part--------------
thread: 2  opecode 14  operand: 121
thread: 1  opecode 12  operand: 120
thread: 2  opecode 16  operand: 103
sleeping for 100000 microsec
thread: 2  opecode 16  operand: 103
sleeping for 200000 microsec
thread: 1  opecode 12  operand: 103
thread: 1  opecode 12  operand: 103
x y g value: 1 -1 6
--------------static part--------------
Function name: ThreadFun1
Function name: ThreadFun2
--------------dynamnic part--------------
thread: 2  opecode 14  operand: 121
thread: 1  opecode 12  operand: 120
thread: 2  opecode 16  operand: 103
sleeping for 100000 microsec
thread: 1  opecode 12  operand: 103
sleeping for 200000 microsec
thread: 2  opecode 16  operand: 103
thread: 1  opecode 12  operand: 103
x y g value: 1 -1 10
```

eecs228 — -bash — 1

```
--------------static part--------------
Function name: ThreadFun1
Function name: ThreadFun2
--------------dynamnic part--------------
thread: 2  opecode 14  operand: 121
thread: 1  opecode 12  operand: 120
sleeping for 100000 microsec
thread: 1  opecode 12  operand: 103
thread: 1  opecode 12  operand: 103
sleeping for 200000 microsec
thread: 2  opecode 16  operand: 103
thread: 2  opecode 16  operand: 103
x y g value: 1 -1 36
--------------static part--------------
Function name: ThreadFun1
Function name: ThreadFun2
--------------dynamnic part--------------
thread: 2  opecode 14  operand: 121
thread: 1  opecode 12  operand: 120
sleeping for 100000 microsec
thread: 2  opecode 16  operand: 103
sleeping for 200000 microsec
thread: 1  opecode 12  operand: 103
thread: 1  opecode 12  operand: 103
thread: 2  opecode 16  operand: 103
x y g value: 1 -1 18
--------------static part--------------
Function name: ThreadFun1
Function name: ThreadFun2
--------------dynamnic part--------------
thread: 2  opecode 14  operand: 121
thread: 1  opecode 12  operand: 120
thread: 1  opecode 12  operand: 103
thread: 2  opecode 16  operand: 103
sleeping for 200000 microsec
thread: 1  opecode 12  operand: 103
thread: 2  opecode 16  operand: 103
x y g value: 1 -1 24
--------------static part--------------
Function name: ThreadFun1
Function name: ThreadFun2
--------------dynamnic part--------------
thread: 2  opecode 14  operand: 121
thread: 1  opecode 12  operand: 120
thread: 2  opecode 16  operand: 103
sleeping for 200000 microsec
thread: 1  opecode 12  operand: 103
thread: 1  opecode 12  operand: 103
thread: 2  opecode 16  operand: 103
x y g value: 1 -1 18
exit normally
```

Thank You