# 本周进度

**2023-07-28**

- 阅读 zord
- 尝试用 zord 跑了几个 demo
- 尝试读 zord 源码，只看懂了一小部分

# 用法

- 编译：按照 readme.txt ，先编译 Z3，然后编译 CBMC ，从 0 开始编译需要 40 分钟左右
- 对于一个 C 程序（.c），首先拿 gcc 把它编译成 .i ，然后就可以用 zord 检验（readme.txt）
- 不编译，可能（语法）解析失败
- 例子：
  - account5（无法解析，原因不详，报错很奇怪）
  - account10（sat，assertion 可能触发）
  - airline7（unsat，assertion 无论如何都不会触发）

```
Run:
Place executable z3 and cbmc together, then run:
./cbmc --32 --no-unwinding-assertions [*.i] --z3
```

- Zord 由 CBMC（前端）和 Z3（后端）组成，CBMC 会生成一个 Z3 语法的临时文件，然后调用 Z3 进程
- CBMC 部分：找到了一些看起来比较关键的代码段，但是并不能完全搞清楚
- Z3 部分：还没开始看

# CBMC 前端

- 使用一种叫做 Symex 的“IR”，一些变量的命名猜不出来

Every GOTO-program instruction has a specific type. You can see the instruction type at **goto_program_instruction_typet** but we will also list some of them here for illustration purposes:

```
enum goto_program_instruction_typet
{
  [...]
  GOTO = 1,               // branch, possibly guarded
  ASSUME = 2,             // assumption
  ASSERT = 3,             // assertions
  SKIP = 5,               // just advance the PC
  SET_RETURN_VALUE = 12, // set function return value (no control-flow change)
  ASSIGN = 13,            // assignment lhs:=rhs
  DECL = 14,              // declare a local variable
  DEAD = 15,              // marks the end-of-live of a local variable
  FUNCTION_CALL = 16,     // call a function
  [...]
};
```

(*NOTE*: The above is for illustration purposes only – the list is not complete, neither is it expected to be kept up-to-date. Please refer to the type **goto_program_instruction_typet** for a list of what the instructions look like at this point in time.)

- 从编译的角度把输入的程序翻译成 goto program，顺带进行一些（可能的）静态分析

# CBMC 前端

- 和文章描述的步骤关系比较大的部分有：
  - src/goto-symex/memory_model_sc.cpp 重载 () 的部分：构建 event_list，set events_ssa_id，添加 RF、WS 和 PO 关系的约束
  - 添加约束：add_constraint

```
27  void memory_model_sct::operator()(symex_target_equationt &equation)
28  {
29      print(8, "Adding SC constraints");
30      build_event_lists(equation);
31      if (equation.exceed_events_limit()) return;
32      build_clock_type(equation);
33      set_events_ssa_id(equation);
34      choice_symbols.clear();
35      read_from(equation);
36      std::cout << equation.SSA_steps.size() << " steps after addressing read_from relations:" << "\n";
37      write_serialization_external(equation);
38      std::cout << equation.SSA_steps.size() << " steps after addressing write sequences relations: " << "\n";
39
40  #if front_deduce_all_fr
41          from_read(equation);
42  #endif
43
44      program_order(equation);
45      std::cout << equation.SSA_steps.size() << " steps after addressing program orders: :" << "\n";
46  }
```

# CBMC 前端

- 和文章描述的步骤关系比较大的部分有：
  - src/cbmc/all_properties.cpp, all_properties()： 和 SMT-Solver 交互的部分
    - 已经有了除 Error Condition（assert）外的所有约束，这一步编码 $\rho_{err}$（goal），然后将这些约束传给 Z3
    - cover_goal 的重载（）的部分：

```
void cover_goalst::operator()()
{
  _iterations=_number_covered=0;
  decision_proceduret::resultt dec_result;
  // We use incremental solving, so need to freeze some variables
  // to prevent them from being eliminated.
  freeze_goal_variables();
  do
  {
    // We want (at least) one of the remaining goals, please!
    _iterations++;
    constraint();
    dec_result=prop_conv.dec_solve();
    switch(dec_result)
    {
    case decision_proceduret::D_UNSATISFIABLE: // DONE
      break;
    case decision_proceduret::D_SATISFIABLE:
      // mark the goals we got
      mark();
      // notify
      assignment();
      break;
    default:
      error("decision procedure has failed");
      return;
    }
  }
  while(dec_result==decision_proceduret::D_SATISFIABLE &&
        number_covered()<size());
}
```

# Z3 的临时文件

- 删去了 src/solvers/smt2/smt2-dec.cpp 的析构函数中的两个 unlink，然后重新编译，可以在 /tmp 下找到产生的中间文件和 Z3 返回的结果文件

# 计划

- 如果可以的话，想要系统地看一下 CBMC 和 Z3，一个星期估计不够用，然后再回来理解源码
- 想尝试看一下 czg 学长现在写了的部分，看看能不能理解
- 没解决的问题（太多了……）：
  - Event graph 是怎么构建和维护的?
  - 需要不断尝试激活 RF、WF 边，搜索的过程在哪？维护 acyclic 的 event graph 部分又在哪?
  - 仅仅看到了 RF、WS、PO，前端并不做 FR 的生成，很想知道
  - minisat 里面有 ICD 算法的实现……