本周进度

2023-08-04

- 继续阅读 zord 源码（没有读 Z3 部分）
- 阅读 cobra 前三节和 czg 学长的毕业设计（algo-detail.pdf），了解可串行化检验问题的背景和现有方法
- 跑通了 czg 学长的实现，初步了解 SER-checker 各部分的实现细节，但并未仔细阅读代码
- 学习 an introduction to SAT and SMT（2/10）

SAT（SMT）如何处理找环的问题?

- 并不太支持在 SAT（SMT）建模中直接编码所有的环（开销太大），往往是采用与 SMT Solver 不断交互的方式
- 例 zord:
  - 编码所有的已知信息（如 PO），同时产生其他关系（RF 和 WS），但并不激活这些边
  - <u>一开始将所有的限制交给 SMT Solver 求解:</u>
    - SMT Solver 很可能会判断出 SAT 的结论，并给出一组可行解
      - 检查这组可行解是否满足无环的性质，若无环，则说明这是一种可行的执行路径，在该路径下 assertion 可能被触发，结论为“验证失败”
      - 若有环，则将这个环所在的信息编码成为 conflict clause ，交给 SMT Solver ，并回到这一步开头
    - 若 SMT Solver 给出 UNSAT 的结论，证明不存在一种执行路径，使 assertion 被触发，结论为“验证成功”

zord 的实现

该过程的骨架在 src/cbmc/bmc.cpp，bmct::run() 中

纠正一些上星期的问题

CBMC 前端：翻译源程序为 IR

src/goto-symex/memory_model_sc.cpp:
构建 event 列表；构建 SSA id；构建 read_from，write_serialization，program_order 关系

上星期汇报的内容

按照 readme.txt 的描述，应该是走的这个分支

src/cbmc/bmc.cpp，bcmt::decide()（654）：
直接用命令行和 Z3 交互，但似乎只有一趟

需加上命令行选项：- - all-properties

src/cbmc/all_properties.cpp（634）：
收集所有 ρ_{err}，并与 minisat 交互

疑问：
1. 为什么只需要与 Z3 交互一趟？
2. Value Assignment Encoding 的实现在哪？

minisat 中实现了 ICD 算法以及 zord 5.4 theory propagation
（包括 unit-edge propagation 和 from-read propagation）
minisat-2.2.1/minisat/core/ICD.cc : icd_sparse() 向 DAG 中加边

论文里面说 Z3 是后端，不会乱说，猜测修改了 Z3，可能一开始写了 minisat，Z3 里面同样实现了这些功能

ICD algorithm

增量式找环（Incremental Cycle Detection）

定义：为每个结点 v 维护一个伪拓扑序（*pseudo-topological ordering* or *level*）$k(v)$，初始为 1；
为每个结点 v 维护出边集合 $out(v) = \{w \mid (v, w)\}$ 和 入边集合 $in(v) = \{u \mid (u, v) \wedge k(v) = k(u)\}$

1. 假设要添加 (v,w) 这条边，若 $k(v) < k(w)$ ，一定无环，goto 4.
2. 从 v 出发进行后向搜索（search backward），只扩展那些满足 $k(u) = k(v)$ 的点 u，并记录访问到的点集 B 。满足以下三条件之一时，停止：
    1. 访问了 w，即找到了一条环
    2. 已经访问了 $\Delta = \min\{m^{\frac{1}{2}}, n^{\frac{2}{3}}\}$ 个点
    3. 所有前驱点都已经访问到了

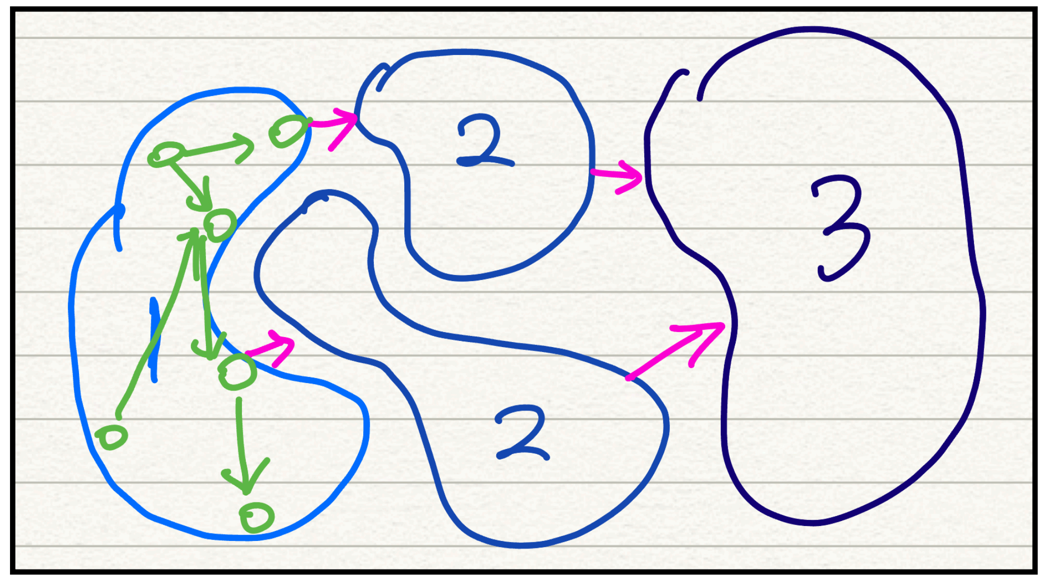
讨论：

    1. 若没有访问 Δ 个结点，并且 $k(w) = k(v)$ ，认为当前 *level* 没有被破坏，goto 4.
    2. 若没有访问 Δ 个结点且 $k(w) < k(v)$ ，设置 $k'(w) = k(v)$
    3. 若访问了 Δ 个结点，设置 $k'(w) = k(v) + 1$，$B' = \{v\}$

3. 从 w 出发进行前向搜索（search forward），假设当前访问到的边为 (x,y)：
    1. 若 $y \in B$ ，找到了环
    2. 若 $k(y) = k(x)$ ，将 x 加入 $in(y)$
    3. 若 $k(y) < k(x)$ ，设置 $k'(y) = k(x)$ ，并扩展 y （递归访问 y 的所有后继）
4. 加入 (v,w) ，若 $k(v) = k(w)$ ，向 $in(w)$ 中加入 v

正确性理解

- 直观感受：把 DAG 分块，块也是小 DAG，每块大小不超过 Δ ，不同的 *level* 代表不同的拓扑序；但相同的 *level* 也不代表同一块（定义可能是：*level* 相同，且有边连接的点集为一块）



- 引理 1：若存在 v 到 w 的路径（path），则 $k(v) \leq k(w)$. $\iff$ $k(v) > k(w) \implies$ 不存在从 v 到 w 的路径
- 所以：在前向搜索的时候不需要扩展那些高 *level* 的结点，一定能找到环（如果有）；
  - 后向搜索不影响正确性，应该只是对搜索空间的一种压缩（降低时间开销）

正确性理解（cont’d）

后向搜索（backward）中的三种情况

讨论：

1. 若没有访问 Δ 个结点，并且 $k(w)=k(v)$，认为当前 *level* 没有被破坏，goto 4.
2. 若没有访问 Δ 个结点且 $k(w)<k(v)$，设置 $k'(w)=k(v)$
3. 若访问了 Δ 个结点，设置 $k'(w)=k(v)+1$，$B'=\{v\}$

3. 从 w 出发进行前向搜索（search forward），假设当前访问到的边为 (x,y)：
   1. 若 $y\in B$，找到了环
   2. 若 $k(y)=k(x)$，将 x 加入 $in(y)$
   3. 若 $k(y)<k(x)$，设置 $k'(y)=k(x)$，并扩展 y（递归访问 y 的所有后继）
4. 加入 (v,w)，若 $k(v)=k(w)$，向 $in(w)$ 中加入 v

- 从 v 开始后向搜索，若：
- 以 v 为边界的块大小不足 Δ，则：
  - 把 w 加入这一块，若 $k(w)=k(v)$，则边 (v,w) 不破坏 DAG 性质
  - 若 $k(w)<k(v)$，则将 w 的 *level* 提升至 $k(v)$，并通过前向搜索尝试找到环并更新 w 所在的这一块
- 以 v 为边界的这一块大小已经超过 Δ，从 w 出发另开一新块（提升 w 的 level，$k(w)=k(v)+1$），并通过设置 $B=\{v\}$来更正前向搜索的范围
- 引理2. w 的 level 提升之后，相当于赋予了走到 v 所在的这一块的资格，前向搜索会找到环

下周计划

- 如果能看懂并且理解 ICD 算法的证明，下周做一个汇报（）
- 继续学 An introduction to SAT and SMT
- 尝试上手改 czg 学长的实现，方向？
- 尝试看 Z3 ，如果可以最好理解 zord 是怎么进行修改的（可能来不及）
- 问题：接下来的方向？

问题

- 在 zord 中，感觉对于 SMT 问题的建模，绝大部分都是 SAT，似乎仅有 Value Assignment Encoding 和 Error Condition 的编码部分才涉及到 SMT；
  - Value Assignment Encoding：

```
void* thr1(void* arg) {
  if(x2 == 1)  m3 = 1;
  else m4 = x3;
   y2  = x4 + 1;
}
```

$$(x_2 = 1 \rightarrow m_3 = 1) \land (\neg(x_2 = 1) \rightarrow m_4 = x_3) \land (y_2 = x_4 + 1)$$

  - Error Condition：

$$\text{assert}(!(m_2 == 1 \;\&\&\; n_2 == 1));$$

$$\rho_{err} := (m_2 = 1) \land (n_2 = 1)$$

- 这两个在数据库的 Serializability 建模中都没有 → Monosat（over booleans and bitvectors）
- 好像就是一个 SAT Solver 能解决的问题，自己写，会更快吗?