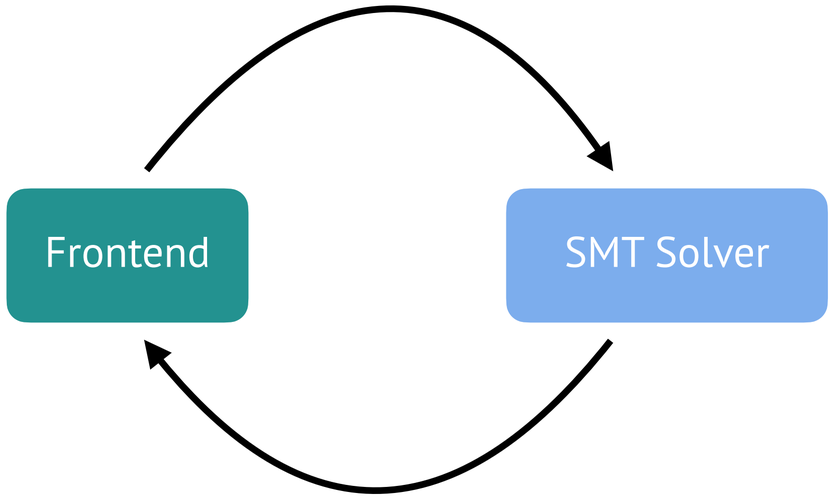
本周进度

2023-08-11

- 看了 A Tutorial on SAT Solving，了解 CDCL、SLS 等常见 SAT 求解策略（对于启发式优化策略的了解比较有限）
- 看了 zord 中 z3 后端的源码，初步了解了框架构成，但对于 z3 提供的搜索算法，还看不懂
- 看了 Monosat 的 PhD thesis 的 2 Background、3 Monotonic Theories、4 Framework、5.1 Solver 和 5.2.2 Acyclicity （略过 5.2.1 Reachability），初步了解了 Monosat 中单调性的应用方法
- 简单浏览了 czg 学长的代码实现，主要为如何使用 z3 进行求解，进行少量修改包括：
  - 基础设施：随手写了个脚本跑了一下现有的历史记录
  - 加入了 AbstractSolver 和 SolverFactory，准备接入不同的 solver

求解的框架

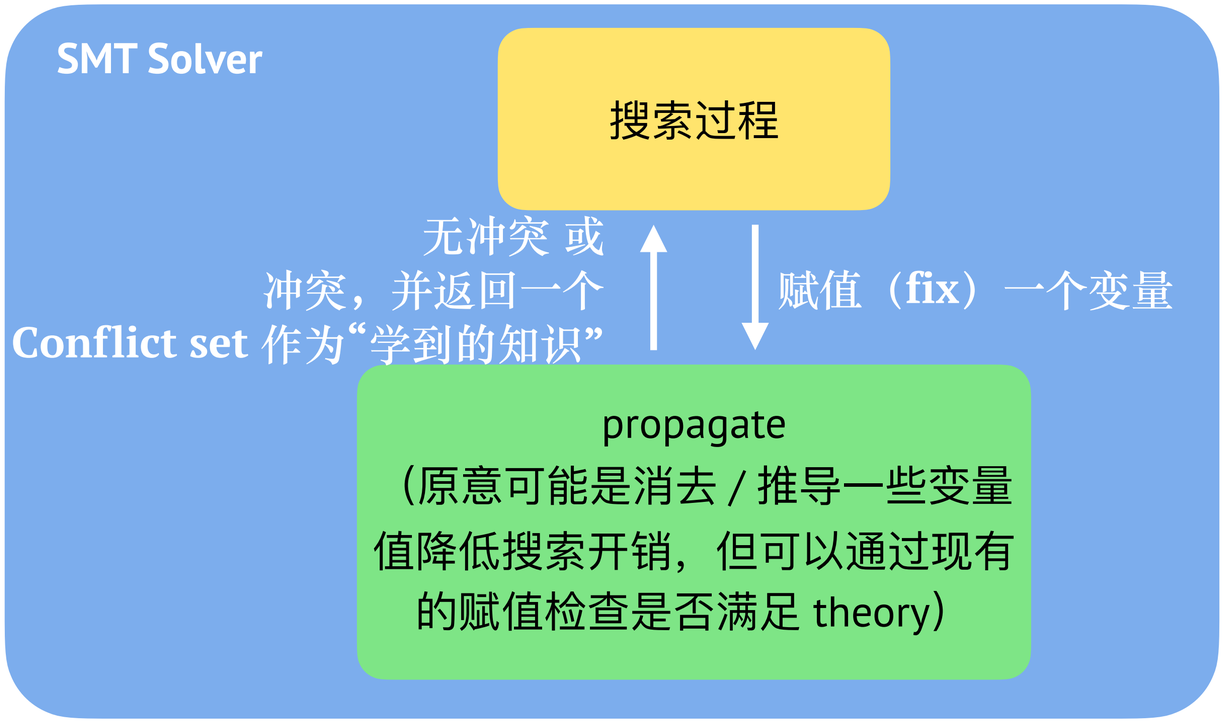
检查性质（如 无环），给出 conflict clause，
再次与求解器交互



给出求解结果

不需要修改 SMT Solver，
把 SMT Solver 作为黑盒使用

把需要维护的性质（如 无环）作为一个新的 theory，
并在求解的过程中辅助推断和验证



一个很好的定义（from monosat-thesis）：定义 theory T
中包含一组需要满足的 formula 的集合，
若模型 $\mathcal{M} \vDash \phi$ under T，则 $\mathcal{M} \vDash T \cup \phi$，
对于 acyclicity：显式写出来的代价非常大，但是可以通过
一些算法快速检验给定的边集构不构成环

z3（作为 zord 的后端）

- 作者实现了一个新的 theory theory_oc：
  - 这里的 theory 是狭义的，指 z3 中的数据结构 theory 的一个实现，类似于 theory_arith 、theory_fpa
  - 核心接口：

```
 73 >   theory_var theory_oc::mk_var(enode * n)…
 89
 90 >   theory_var theory_oc::mk_var(enode * n, symbol s)…
117
118 >   bool theory_oc::internalize_atom(app * atom, bool gate_ctx)…
179
180 >   void theory_oc::assign_eh(bool_var v, bool is_true)…
221
222 >   void theory_oc::propagate()…
287
288 >   void theory_oc::push_scope_eh() {…
292
293 >   void theory_oc::pop_scope_eh(unsigned num_scopes) {…
```

似乎和初始化和变量的 equality 有关，具体不明白

fix 一个变量以后进行的操作

推断 / 检查有无环，并寻找 conflict set

搜索过程中状态的保存和恢复

theory_oc

仿照 monosat-thesis 中 P16 的定义，尝试给出 theory oc 的 signature 的定义

- Sorts = {Oc}，即 $(|x_i|)^r$

```
(declare-fun |c::numOfTickets#1$wclk| () Oc)
```

- Predicates = { $\prec_{po}$, $\prec_{rf}$, $\prec_{wf}$, $\prec_{fr}$ } ， 连接两个 Oc，表示一种先序关系

```
(declare-fun |c::numOfTickets#1$wclk| () Oc)
; find_symbols
(declare-fun |c::numOfTickets#2$wclk| () Oc)
; set_to true
(assert (or (not |coi535|) (oclt-coi |c::numOfTickets#1$wclk| |c::numOfTickets#2$wclk| coi535)))
```

- Functions $= \varnothing$
- Formulas： 1、fr propagation $rf^x_{j,i} \wedge ws^x_{j,k} \rightarrow (|x_i|)^r \prec_{fr} (|x_k|)^w$　　2、acyclicity

z3: theory.h

- 换言之，只要实现了 z3 中 theory.h 的接口，在不了解搜索过程的前提下也可以写出用户自定义的 theory
- 按理来说，有了这些接口，应该可以理出一个框架，把这些接口填进去
- 然而看不懂，部分能看懂的框架在 smt_context.cpp 和 smt_consequences.cpp 中，搜索过程过于复杂，技巧性太强
- 一个（主要）困难：**缺乏指导文档**，我能找到的文档要么简易讲解原理，要么讲解高级语言 API ，我没有找到对于 z3 中数据结构和框架的（详细）指导

z3 中另一套用户接口

即 czg 学长现有的实现

- 示例文档（较简略，python）：https://microsoft.github.io/z3guide/programming/Example%20Programs/User%20Propagator/
- 有个 paper User-Propagation for Custom Theories in SMT Solving;
- 附带了一个 ppt
- 粗略浏览，觉得对于编程价值不大，未详细看

Z3 为领域相关求解器提供了 `z3::user_propagator_base` 基类作为接口，通过覆写接口方法的方式在 SMT 求解过程中注册回调函数。该接口主要有 `push`、`pop` 与 `fixed` 三个方法组成。

SMT 单调性理论

- 其核心思想在于维护搜索的上下界，利用单调性剪枝
- 但是感觉对于 partial order 的拓展比较勉强
- 一个问题：Chapter 4. algorithm 6 中，这个是笔误吗?

Algorithm 6 Conflict set generation for finite monotonic theories. *Assuming all predicates are positive monotonic.* Algorithm 6 takes $\mathcal{M}$, a (partial) assignment that is unsatisfiable in T, and returns a conflict set, a subset of the atoms assigned in $\mathcal{M}$ that are mutually unsatisfiable in T.

function *T-Analyze*($\mathcal{M}$)
  for each Variable x of sort σ **do**
    Compute x^-, x^+ as in *T-Propagate*.
    case $(x < \sigma_\perp) \in \mathcal{M}$
      return $\{(x < \sigma_\perp)\}$
    case $(x > \sigma_\perp) \in \mathcal{M}$
      return $\{(x > \sigma_\perp)\}$
    case $\nexists \sigma_i : (\sigma_i \geq x^-) \wedge (\sigma_i \leq x^+)$
      return $\{(x \geq x^-), (x \leq x^+)\}$
  for each monotonic predicate atom $p(t_0, t_1, \ldots)$ **do**
    case $\neg p \in \mathcal{M}$, $evaluate(p(x_0^-, x_1^-, \ldots)) \mapsto$ TRUE
      return $\{\neg p, (x_0 \geq x_0^-), (x_1 \geq x_1^-), \ldots\}$
    case $p \in \mathcal{M}$, $evaluate(p(x_0^+, x_1^+, \ldots)) \mapsto$ FALSE
      return $\{p, (x_0 \leq x_0^+), (x_1 \leq x_1^+), \ldots\}$

设想

- 先用 monosat 试试吧……
- 我觉得（以下论断均为随口胡说，没有经过实验验证）
  - 1、monosat 的引擎可能较老
  - 2、acyclicity 是很简单的问题，不需要用到 bitvector 等高级的抽象，monosat 仍然有冗余
  - 3、对于事务的可串行化检查问题，其状态空间为 $2^{|C|}$ 更为密集，或许不应该将其一般化为一个在图中加边的问题，在搜索过程中可能有更加激进的策略
  - z3 中的两套用户接口可能都可以支持单调性理论中 T-propagate 和 T-analyze 的实现，因此可以用 z3 实现单调性理论（我偏向于写 theory，这样容易前后端分离）

有关 SER-checker 的代码

- 写了一个最简单的 python 脚本，用于跑现有的 history，现有结果：

name	sessions	txns	events	constrains	construct time	solve time	accept
10_45_15_1000	11	451	7749	6236	113ms	25788ms	true
15_60_15_1000	16	901	14500	23616	371ms	197275ms	true
15_45_25_1000	16	676	17875	33152	701ms	317865ms	true
15_100_15_1000	16	1501	23500	62430	1404ms	2035807ms	true
15_15_15_1000	16	226	4347	1492	36ms	2671ms	true
15_45_15_500	16	676	10625	24835	420ms	190975ms	true
15_45_15_1000	16	676	11125	12755	205ms	80506ms	true
15_75_15_1000	16	1126	17875	35045	593ms	370903ms	true
15_45_15_1250	16	676	11375	10629	155ms	60180ms	true
15_45_5_1000	16	676	4341	2143	39ms	6515ms	true
15_45_15_1500	16	676	11624	8786	156ms	42546ms	true
15_30_15_1000	16	451	7748	5863	99ms	23634ms	true
25_45_15_1000	26	1126	17875	35590	592ms	379598ms	true
20_45_15_1000	21	901	14500	22742	378ms	171094ms	true
5_45_15_1000	6	226	4343	1507	40ms	2552ms	true
15_45_15_750	16	676	10875	16796	285ms	103443ms	true
15_45_20_1000	16	676	14500	22020	392ms	159005ms	true
15_45_10_1000	16	676	7750	6337	110ms	28679ms	true

with - - pruning

name	sessions	txns	events	constrains	construct time	solve time	accept
15_400_15_1000	16	6001	91000	982558	72138ms	204880ms	true
10_45_15_1000	11	451	7749	6236	114ms	699ms	true
15_60_15_1000	16	901	14500	23616	407ms	3166ms	true
15_45_25_1000	16	676	17875	33152	679ms	2134ms	true
15_100_15_1000	16	1501	23500	62430	1301ms	8310ms	true
15_15_15_1000	16	226	4347	1492	32ms	164ms	true
15_45_15_500	16	676	10625	24835	445ms	1936ms	true
15_45_15_1000	16	676	11125	12755	209ms	1492ms	true
15_75_15_1000	16	1126	17875	35045	646ms	5032ms	true
15_45_15_1250	16	676	11375	10629	178ms	1585ms	true
15_45_5_1000	16	676	4341	2143	42ms	1338ms	true
15_45_15_1500	16	676	11624	8786	162ms	1511ms	true
15_30_15_1000	16	451	7748	5863	101ms	672ms	true
25_45_15_1000	26	1126	17875	35590	735ms	4338ms	true
20_45_15_1000	21	901	14500	22742	405ms	2868ms	true
15_200_15_1000	16	3001	46000	247367	8623ms	39864ms	true
5_45_15_1000	6	226	4343	1507	34ms	180ms	true
15_300_15_1000	16	4501	68500	559672	30106ms	104479ms	true
15_45_15_750	16	676	10875	16796	294ms	1675ms	true
15_45_20_1000	16	676	14500	22020	429ms	1877ms	true
15_45_10_1000	16	676	7750	6337	113ms	1376ms	true

- 问题：为啥会（应该在 z3 里面被） killed？加上了 pruning 之后为什么可以求解 history 的变多了？为啥没有不可串行化的例子？

初步计划

- 给现有的 SER-checker 换后端（已经把接口写好了）：
  - 先直接试试 monosat
  - 用 z3 实现一下单调性理论，比对一下效果
  - 应该还是需要手写 CDCL（T），但是这是一个需要长期积累的过程
- 这种二选一的问题可能还有别的性质可以挖掘

- 感想：czg 学长的实现太强了，我应该写不出这么好的代码……

有关 TLA+ 和 CC 的问题

- CC 看起来是原来协议的一个 Safe Abstraction，为什么说利用 CC 性质编写 TLA+ 检查需要修改 TLC 呢?
- 贺老师问的问题：在 TLC 失败之后，有没有尝试过使用别的形式化验证工具？是怎么考虑的?
- 一个灵魂问题：为什么当时想做形式化验证？为什么选择 TLA+ 处于哪些考虑?