Introduction to the CVC5 Solver

CVC5求解器介绍

2025年秋季数理逻辑大作业，研究生组

陈泓臻　25031111097

目录/CONTENTS

CVC5求解器背景简介

Part .01

背景简介

CVC5求解器是一款先进的可满足性模理论(SMT)求解器，为该系列的第五个工具，其前身为CVC4。

随着近年来硬件验证复杂度的增加与针对软件的形式化验证的普及，SMT求解的需求也随之增加(如浮点数、字符串等新理论的支持扩展需求)，加之CVC4求解器的局限性较大，在众多因素的推动下CVC5求解器随之诞生。

CVC5求解器由斯坦福大学、哥伦比亚大学与爱荷华大学等机构牵头开发，主要用于形式验证、程序分析、软件测试等领域。

CVC5求解器原理简介

Part .02

原理简介：DPLL(T)框架

总体而言，CVC5求解器采用分层推理引擎设计，遵循正确性优先、模块化扩展、理论组合灵活的原则。其核心是DPLL(T)框架，将复杂逻辑问题分解为命题逻辑层和理论推理层的协作求解。

DPLL(T)框架的基本流程：

布尔抽象化

将一阶逻辑公式转换为等价的命题逻辑公式

命题层求解

使用CDCL SAT求解器处理布尔结构

理论一致性检查

理论求解器验证命题赋值的理论可行性

冲突分析与学习

理论冲突反馈到命题层，添加冲突子句

迭代优化

通过理论传播提前发现矛盾，减少搜索空间

原理简介:命题逻辑层

基于CDCL的求解引擎:

布尔约束传播

基于单元传播规则快速赋值

冲突驱动子句学习

从冲突中提取新的约束条件

启发式决策

VSIDS变量选择策略优化搜索顺序

重启与遗忘

避免局部最优，管理学习子句数量

原理简介:理论求解框架

CVC5求解器采用Nelson-Oppen方法处理混合理论，包含:

- **理论分离**: 将混合约束分解为纯理论子问题。
- **相等性传播**: 在理论间传递共享变量的相等信息。
- **迭代协调**: 进行迭代直到所有理论达成一致或无解。

CVC5求解器中使用的其他理论求解器包括算数理论求解器，位向量理论求解器，字符串理论求解器，数组理论求解器，未解释函数理论求解器等，拥有较为强大的理论扩展支持能力。

求解器的使用

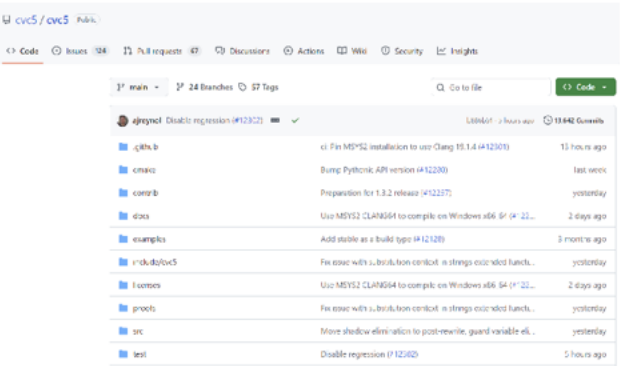
Part .03

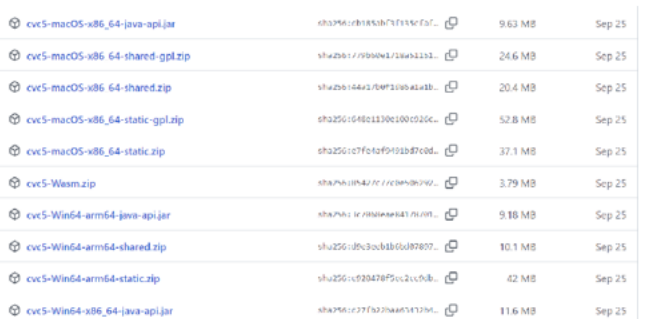
获取与安装

CVC5求解器可以从其github主页获取，链接为https://github.com/cvc5。

求解器的安装有多种方式：

- 首先，可以下载该项目的源码，完成后手动进行原生编译构建。
- 为方便考虑，该求解器也提供了预编译的版本，适配Linux,MacOS,Windows等常见系统，在解压后直接运行其中的exe文件即可使用。





使用示例

在安装完成后，以适用于Windows的预编译版本为例，使用命令行来完成相关使用操作。

首先进入cvc5求解器的exe文件所在路径，可以通过--version与--help等指令查看求解器的相关信息。

也可通过 --show-config指令来查看一些常用的命令参数。

```
cvc5 options:
Most commonly-used cvc5 options:
  --incremental | -i      enable incremental solving [*]
  --lang=LANG | --input-language=LANG | -L LANG
                          force input language (default is "auto"; see --lang
                          help)
  --output=TAG | -o TAG   Enable output tag.
  --parse-only            exit after parsing input [*]
  --preprocess-only       exit after preprocessing input [*]
  --quiet | -q            decrease verbosity (may be repeated)
  --rlimit=N              set resource limit
  --rlimit-per=N | --reproducible-resource-limit=N
                          set resource limit per query
  --stats                 give statistics on exit [*]
  --stats-all             print unchanged (defaulted) statistics as well [*]
  --stats-internal        print internal (non-public) statistics as well [*]
  --tlimit=MS             set time limit in milliseconds of wall clock time
  --tlimit-per=MS         set time limit per query in milliseconds
  --verbose | -v          increase verbosity (may be repeated)
  --verbosity=N           the verbosity level of cvc5
  --decision=MODE | --decision-mode=MODE
                          choose decision mode, see --decision=help
  --copyright             show cvc5 copyright information
  --early-exit            do not run destructors at exit; default on except in
                          debug builds [*]
  --help | -h             full command line reference
  --help-option-categories
                          Print summary of options for each category
```

```
version       : 1.3.2.dev+main@a137ead
scm           : git a137ead on branch main

library       : 1.3.2.dev+main@a137ead

safe-mode     : no
stable-mode   : no
debug code    : no
statistics    : yes
tracing       : no
muzzled       : no
assertions    : no
coverage      : no
profiling     : no
asan          : no
ubsan         : no
tsan          : no
competition   : no
portfolio     : no

cln           : no
glpk          : no
cryptominisat : no
gmp           : yes
kissat        : no
poly          : yes
cocoa         : no
editline      : no
```

使用示例

在具体使用过程中，可以通过命令行来完成对求解器的使用流程。

首先，构建一个简单的问题示例，如找出是否存在三个不等的正整数并使其和为10。对问题建模并且以CVC5需求的smt-lib格式写为smt文件结果如下：

```
(set-option :produce-models true)
(set-logic QF_LIA)

; 谜题：找到三个不同的正整数
; x + y + z = 10
; x < y < z
; x, y, z都是正整数

(declare-const x Int)
(declare-const y Int)
(declare-const z Int)

; 正整数约束
(assert (> x 0))
(assert (> y 0))
(assert (> z 0))

; 和为10
(assert (= (+ x y z) 10))

; 递增顺序
(assert (< x y))
(assert (< y z))

; 所有数字不同
(assert (distinct x y z))

(check-sat)
(get-model)
(get-value (x y z))
```

由于直接运行exe文件会进入交互模式，采用以下命令来对指定路径的smt文件进行求解，命令及结果如下：

```
PS C:\Users\11967\Desktop\TASK\tools\cvc5-Win64-x86_64-static> .\cvc5.exe example1.smt2
sat
(
(define-fun x () Int 2)
(define-fun y () Int 3)
(define-fun z () Int 5)
)
((x 2) (y 3) (z 5))
```

可以看到，cvc5求解器除了问题是否可满足之外还提供了满足时的具体实例。使用如下格式指令还可以实现求解结果的快速保存。

```
.\cvc5.exe example1.smt2 > output.txt
```

Pyhton API调用

Part .04

CVC5的Python API调用

CVC5求解器提供了完整的Pyhthon绑定，有多种方法可以实现在Python中直接使用求解器，此处说明其中较为简单的方法，即直接使用pip将CVC5作为库进行安装。

完成安装后直接使用Pyhthon进行import操作即可。以一个简单的布尔变量问题为例，在Python中的使用如图。

其中各参数，setLogic中"QF_UF"指无量词"未解释函数"，该参数适用于布尔逻辑与命题逻辑问题。其余为一些预先设置，如启用模型生成（即为可满足的公式找到一组具体的赋值或状态）。

```
(base) C:\Users\11967>pip install cvc5
Defaulting to user installation because normal site-packages is not writeable
Collecting cvc5
  Downloading cvc5-1.3.1-cp313-cp313-win_amd64.whl.metadata (761 bytes)
Downloading cvc5-1.3.1-cp313-cp313-win_amd64.whl (13.5 MB)
   ---------------------------------------- 13.5/13.5 MB 9.6 MB/s eta 0:00:00
```

```
[3]: import cvc5
     from cvc5 import Kind

     solver = cvc5.Solver()
     solver.setLogic("QF_UF")
     solver.setOption("produce-models", "true")
     solver.setOption("incremental", "true")
     solver.setOption("stats", "true")

     # 布尔类型可作为变量
     bool_sort = solver.getBooleanSort()

     # 创建3个布尔变量
     A = solver.mkConst(bool_sort, "A")
     B = solver.mkConst(bool_sort, "B")
     C = solver.mkConst(bool_sort, "C")

     # 添加约束: A AND B => C, A=True, B=True
     solver.assertFormula(solver.mkTerm(Kind.IMPLIES,
         solver.mkTerm(Kind.AND, A, B),
         C))
     solver.assertFormula(A)
     solver.assertFormula(B)

     result = solver.checkSat()
     print(f"可满足性: {result}")

     if result.isSat():
         print(f"A = {solver.getValue(A)}")
         print(f"B = {solver.getValue(B)}")
         print(f"C = {solver.getValue(C)}")

     可满足性: sat
     A = true
     B = true
     C = true
```

CVC5的Python API调用

另一方面，Python还提供了另一个较为直观更贴合Python语言习惯的API接口，称为cvc5.pythonic。

与传统API相比，该API的语法格式相对而言更加简洁。其差异性可以由同一问题的变量定义和约束添加清晰的感受到。

```python
import cvc5
from cvc5 import Kind

solver = cvc5.Solver()
solver.setLogic("QF_LIA")
int_sort = solver.getIntegerSort()
x = solver.mkConst(int_sort, "x")
solver.assertFormula(solver.mkTerm(Kind.GT, x, solver.mkInteger(0)))
```

传统API

```python
from cvc5.pythonic import *

solver = Solver()
x = Int('x')
solver.add(x > 0)  # 直接用 > 运算符!
```

Pythonic API

感谢指导