实验三 学习SMT约束求解器

一 实验目的：以z3为例，学习使用约束求解器

二 实验内容：

1. 掌握约束文件的编写方法
2. 能够使用约束求解器完成约束文件的求解

三 实验过程

1. 什么是SMT问题？

\*包含量词、算数运算、比较运算、位运算等的约束叫做SMT问题。**SMT公式的基本构建单元就是常量和函数，常量就是没有参数的函数，所以，基本的构建单元事实上都是函数。**

1. 什么是SMT求解器？

\* SMT求解器是求解SMT问题的算法。它以一个约束文件作为输入，若公式可以满足，求解器可以给出一组满足的解。

1. 什么是Z3？

\* z3是由微软公司开发的一个优秀的SMT求解器，它的输入格式满足SMT-LIB标准，它能够检查逻辑表达式的可满足性。

1. SMT-LIB标准的约束文件编写规范与使用z3求解方法

示例1：布尔表达式运算

（1）代码解析

//set-option：配置z3求解器，来判断输入文件中的命令是否有错误

(set-option :print-success true)

//识别主要逻辑片段(个人理解类似于头文件，但不是必须标明）（QF\_UF：Unquantified formulas built over a signature of uninterpreted (i.e., free) sort and function symbols.）

(set-logic QF\_UF)

//declare-const:申明一个给定类型的常量

(declare-const p Bool)

//assert:添加一个公式到z3的栈中

(assert (and p (not p))) //~p v p =0

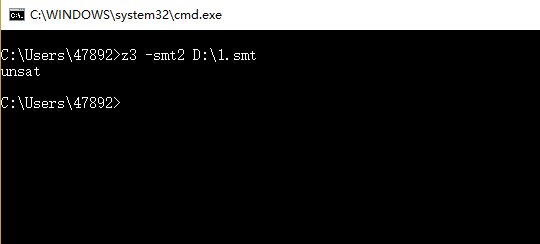
//check-sat:开始求解当前z3栈中公式的可满足性，不满足返回unsat

(check-sat)

//退出

(exit)

1. z3求解（1.smt中包含以上代码）



示例二：求值

（1）代码解析

(set-option :print-success true)

//打印model

(set-option :produce-models true)

//QF\_LIA:Unquantified linear integer arithmetic. In essence, Boolean combinations of inequations between linear polynomials over integer variables.

(set-logic QF\_LIA)

(declare-const x Int)

(declare-const y Int)

//2\*y+x=20

(assert (= (+ x (\* 2 y)) 20))

//x-y=2

(assert (= (- x y) 2))

//是否满足

(check-sat)

//获取一组满足的xy值

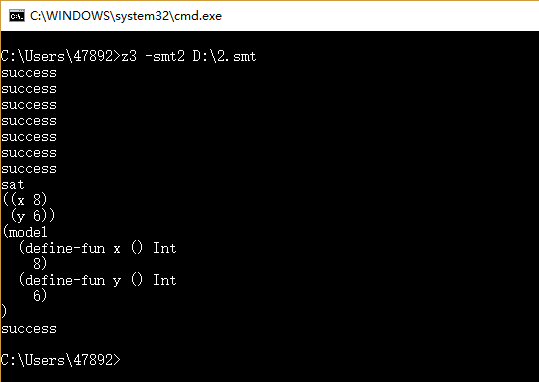
(get-value (x y))

//produce-models true

(get-model)

(exit)

（2）z3求解



示例三：求满足条件的函数集

1. 代码解析

//QF\_UFLIA:Unquantified linear integer arithmetic with uninterpreted sort and function symbols.

(set-logic QF\_UFLIA)

//model:对公式集的一个解释

(set-option :produce-models true)

//declare-fun:申明一个函数，常量相当于无参函数

//x (Int) Int —> int x(int)

(declare-fun x (Int) Int)

(declare-fun y (Int) Int)

(declare-fun t (Int) Int)

//t(0)=x(0),y(1)=t(0),x(1)=y(1)

(assert (= (t 0) (x 0)))

(assert (= (y 1) (t 0)))

(assert (= (x 1) (y 1)))

//~(x(1)=y(0)&y(1)=x(0))

(assert (not

(and (= (x 1) (y 0))

(= (y 1) (x 0)))))

(check-sat)

//获取函数解

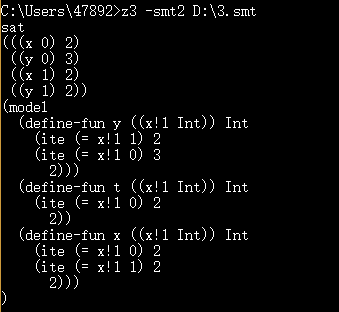
(get-value ((x 0) (y 0) (x 1) (y 1)))

(get-model)

(exit)

1. z3求解

\*get-model获取到的一个公式集解释为



Int y(int x1)

{

if(x1==1) return 2;

if(x1==0) return 3;

else return 2;

}

其他函数同理

示例四：位向量的运算

1. 代码解析

//QF\_BV:Closed quantifier-free formulas over the theory of fixed-size bitvectors.

(set-logic QF\_BV)

(set-option :produce-models true)

//定义常量：32位的位向量

(declare-const x\_0 (\_ BitVec 32))

(declare-const x\_1 (\_ BitVec 32))

(declare-const x\_2 (\_ BitVec 32))

(declare-const y\_0 (\_ BitVec 32))

(declare-const y\_1 (\_ BitVec 32))

//x\_0+y\_0 = x\_1,x\_1-y\_0 = y\_1,x\_1-y\_1 = x\_2 —> (x\_2 = y\_0)&(y\_1 = x\_0)

(assert (= x\_1 (bvadd x\_0 y\_0)))

(assert (= y\_1 (bvsub x\_1 y\_0)))

(assert (= x\_2 (bvsub x\_1 y\_1)))

// ~((x\_2 = y\_0)&(y\_1 = x\_0))

(assert (not

(and (= x\_2 y\_0)

(= y\_1 x\_0))))

(check-sat)

(exit)

1. Z3求解



示例五：解决相似问题

（1）代码解析

(set-option :print-success false)

(set-logic QF\_LIA)

//int x,int y

(declare-const x Int)

(declare-const y Int)

//2\*y+x=20

(assert (= (+ x (\* 2 y)) 20))

//解决相似问题时，可以复用一些定义和断言（以下将x-y=2 放入栈中）

(push 1)

(assert (= (- x y) 2))

(check-sat)

(get-value (x y))

(pop 1)//将x-y=2出栈

//将x-y=3入栈

(push 1)

(assert (= (- x y) 3))

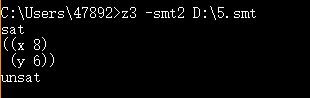
(check-sat)

(get-value (x y))

(pop 1)//x-y=3出栈

(exit)

（2）Z3求解



2\*y+x=20 x-y=3

2\*y+x=20 x-y=2

示例六：无解释类型

1. 代码解析

(set-option :print-success false)

(set-logic QF\_UF)

//declare-sort:定义无解释的类型A（最大限度自由，允许任何和约束一致的解释）

(declare-sort A)

//定义：A a，A b，A c，A d，A e

(declare-const a A) (declare-const b A) (declare-const c A)

(declare-const d A) (declare-const e A)

//a=c | b=c, d=a | d=b, e=a | e=b

(assert (or (= c a)(= c b)))

(assert (or (= d a)(= d b)))

(assert (or (= e a)(= e b)))

(push 1)

//c!=d

(assert (distinct c d))

(check-sat)

(pop 1)

(push 1)

//c!=d but d = e

(assert (distinct c d e))

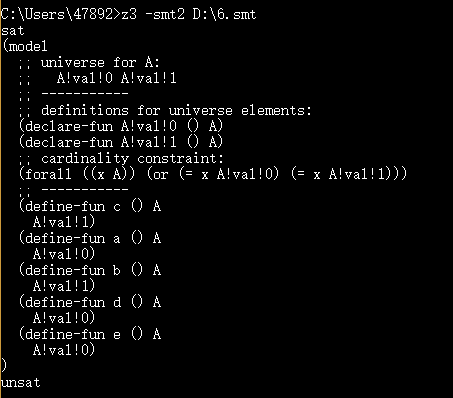
(check-sat)

(pop 1)

(exit)

（2）Z3求解

模型中为A给定了一个抽象的值



示例七：表达式命名

1. 代码解析

(set-option :produce-assignments true)

(set-logic QF\_UF)

//bool p,bool q,bool r

(declare-const p Bool) (declare-const q Bool) (declare-const r Bool)

//P：!p , PQ：!p and q，R：!R

// not（！(!p and q）=（！r））

(assert (not (=(! (and (! p :named P) q) :named PQ) (! r :named R))))

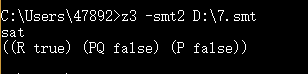
(check-sat)

//输出代表指定表达式的bool值

(get-assignment)

(exit)

（2）Z3求解



示例八：输出unsat-cores

1. 代码解析

(set-option :produce-unsat-cores true)

(set-logic QF\_UF)

//声明bool常量

(declare-const p Bool) (declare-const q Bool) (declare-const r Bool)

(declare-const s Bool) (declare-const t Bool)

//p蕴含q的非

(assert (! (=> p q) :named PQ))

//q蕴含r的非

(assert (! (=> q r) :named QR))

//r蕴含s的非

(assert (! (=> r s) :named RS))

//s蕴含t的非

(assert (! (=> s t) :named ST))

//q不蕴含s的非

(assert (! (not (=> q s)) :named NQS))

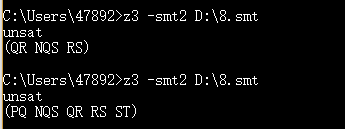
(check-sat)

//输出不能满足的该公式中不合要求的地方

(get-unsat-core)

(exit)

（2）Z3求解



将倒数第四句改为(assert (! (not (=> p t)) :named NQS))

示例九：获得assertion栈内信息

1. 代码解析

//使用报错，提示替换为(set-option :interactive-mode true)

(set-option :produce-assertions true)

(set-logic QF\_UF)

(declare-const p Bool) (declare-const q Bool)

(push 1)

//p or q入栈

(assert (or p q))

(push 1)

//！p入栈

(assert (not q))

//获得栈顶元素:p or q

(get-assertions)

(pop 1)

//获得栈顶元素:!p

(get-assertions)

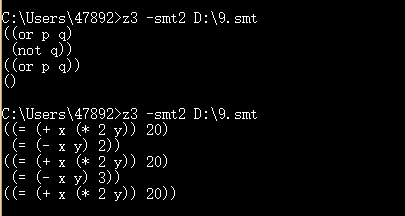
(pop 1)

//获得栈顶元素：()

(get-assertions)

(exit)

（2）Z3求解



又使用了示例五公式测试

五 实验收获

通过学习官网上给出的典型示例，通过举一反三基本掌握了smt-lib标准的约束文法，学会了利用z3进行目标求解。