



### 介绍

Z3 是微软研究谚语的一个最先进的理论。它可以用来检查逻辑公式的满足在一个或多个理论。Z3 为软件分析和验证工具提供了令人信服的匹配, 因为一些通用软件直接映射到支持的理论中。

### 基本命令

Z3 输入格式是由[SMT-LIB 2.0 标准](http://www.smtlib.org/" \t "https://www.rise4fun.com/Z3/tutorialcontent/_blank)定义的一个扩展。Z3 脚本是一系列命令。命令**echo**显示一条消息（相当于注释）。在内部, Z3 维护一堆用户提供的公式和声明。我们说这些是用户提供的**assertions (断言)**。命令**declare-const**声明给定类型的常量 (又称排序)。命令**declare-fun** 声明一个函数。在下面的示例中, 我们声明了一个接收整数和布尔值的函数, 并返回一个整数。

(echo "starting Z3...")  
(declare-const a Int)  
(declare-fun f (Int Bool) Int)

命令**assert**将公式添加到 Z3 内部堆栈中。我们说, 如果有一个解释 (对于用户声明的常量和函数), 则 Z3 堆栈中的公式集是可满足的, 这使得所有断言的公式都是真的。

(declare-const a Int)  
(declare-fun f (Int Bool) Int)  
(assert (> a 10))  
(assert (< (f a true) 100))  
(check-sat)

第一个断言公式表示常数a必须大于10。第二种情况下，应用于a和真值的函数f必须返回小于100的值。命令校验SAT确定Z3栈上当前公式是否满足。如果公式是可满足的，则Z3返回**sat**，如果它们不可满足，则Z3返回**unsat**。当不能确定公式是否满足时，Z3也可以返回**未知量**。

当命令检查SAT返回**sat**时，命令**get-model**可以用来检索使得Z3内部堆栈上所有公式都正确的解释。

(declare-const a Int)  
(declare-fun f (Int Bool) Int)  
(assert (> a 10))  
(assert (< (f a true) 100))  
(check-sat)  
(get-model)

解释采用了定义。命令**define-fun**表示定义函数，例如，定义：说明模型中a值是[Val]。

(define-fun a () Int [val])

定义：与编程语言中使用的函数定义非常相似。在这个例子中，**x1**(即**x!1**)和 **x2**(即**x!2**)是由Z3创建的函数解释的参数。对于这个简单的例子，f的定义是基于命令**ite**的 (又名 **if-then-elses** or 条件表达式)。

(define-fun f ((x!1 Int) (x!2 Bool)) Int  
 ...  
)

例如，表达式：当x1=11且x2为false时，返回21。否则，返回0。

(ite (and (= x!1 11) (= x!2 false)) 21 0)

### 使用作用域

在一些应用中，我们想探讨几个类似的问题，它们共享几个定义和断言。我们可以使用命令**push**和**pop**来做这件事。Z3 维护**declarations**和**assertions**的全局堆栈。命令**push**通过保存当前堆栈大小创建新的作用域。命令**pop**删除它与匹配推送之间执行的任何断言或声明。**check-sat**和**get-assertions** 命令总是操作在全局堆栈的内容上。

在下面的示例中，命令 **(assert p)** 由于**pop**命令删除了p的声明而出现错误。如果最后一个**pop**命令被删除，则错误被纠正。

(declare-const x Int)  
(declare-const y Int)  
(declare-const z Int)  
(push)  
(assert (= (+ x y) 10))  
(assert (= (+ x (\* 2 y)) 20))  
(check-sat)  
(pop) ; remove the two assertions  
(push)   
(assert (= (+ (\* 3 x) y) 10))  
(assert (= (+ (\* 2 x) (\* 2 y)) 21))  
(check-sat)  
(declare-const p Bool)  
(pop)  
(assert p) ; error, since declaration of p was removed from the stack

**push**和**pop**命令可以可选地接收SMT 2语言指定的数字参数。

### 配置

命令**set-option**用于配置Z3。Z3有几个选项来控制它的行为。这些选项中的一些（例如：**produce-proofs**）只能在任何声明或断言之前设置。我们使用命令**reset**删除所有断言和声明。复位命令之后，可以设置所有配置选项。

**[load in editor](https://www.rise4fun.com/Z3/0nSM?frame=1&menu=0&course=1" \t "https://www.rise4fun.com/Z3/tutorialcontent/default)**

(set-option :print-success true)   
(set-option :produce-unsat-cores true) ; enable generation of unsat cores  
(set-option :produce-models true) ; enable model generation  
(set-option :produce-proofs true) ; enable proof generation  
(declare-const x Int)  
(set-option :produce-proofs false) ; error, cannot change this option after a declaration or assertion  
(echo "before reset")  
(reset)  
(set-option :produce-proofs false) ; ok

当使用Z3被另一个应用程序控制时，选项**print-success true**真是特别有用的。在这种模式下，否则打印不出任何输出的命令将打印success。

### 其他命令

命令 **(display t)**只是将Z3漂亮的打印机应用于给定的表达式。命令**(simplify t)** 显示一个可能更简单的等价于t的表达式。这个命令接受许多不同的选项，（帮助简化）将显示所有可用的选项。

**[load in editor](https://www.rise4fun.com/Z3/dy3?frame=1&menu=0&course=1" \t "https://www.rise4fun.com/Z3/tutorialcontent/default)**

(declare-const a (Array Int Int))  
(declare-const x Int)  
(declare-const y Int)  
(display (+ x 2 x 1))  
(simplify (+ x 2 x 1))  
(simplify (\* (+ x y) (+ x y)))  
(simplify (\* (+ x y) (+ x y)) :som true) ; put all expressions in sum-of-monomials form.  
(simplify (= x (+ y 2)) :arith-lhs true)  
(simplify (= (store (store a 1 2) 4 3)  
 (store (store a 4 3) 1 2)))  
(simplify (= (store (store a 1 2) 4 3)  
 (store (store a 4 3) 1 2))  
 :sort-store true)  
(help simplify)

**define-sort** 命令定义了一个新的排序符号，它是一个排序表达式的缩写。新的排序符号可以被参数化，在这种情况下，参数的名称在命令中指定，排序表达式使用排序参数。命令的形式是：

(define-sort [symbol] ([symbol]+) [sort])

下面的示例定义了用于排序表达式的几个缩写。

**[load in editor](https://www.rise4fun.com/Z3/0WB?frame=1&menu=0&course=1" \t "https://www.rise4fun.com/Z3/tutorialcontent/default)**

(define-sort Set (T) (Array T Bool))  
(define-sort IList () (List Int))  
(define-sort List-Set (T) (Array (List T) Bool))  
(define-sort I () Int)  
  
(declare-const s1 (Set I))  
(declare-const s2 (List-Set Int))  
(declare-const a I)  
(declare-const l IList)  
  
(assert (= (select s1 a) true))  
(assert (= (select s2 l) false))  
(check-sat)  
(get-model)

### 命题逻辑

(declare-const p Bool)  
(declare-const q Bool)  
(declare-const r Bool)  
(define-fun conjecture () Bool  
 (=> (and (=> p q) (=> q r))  
 (=> p r)))  
(assert (not conjecture))  
(check-sat)