Z3

Hartur Barreto Brito
Escola Politécnica de Pernambuco

2015.1

1 Introdução

1.1 Definição

Z3 é um provador de teoremas (SMT) de alta performance desenvolvido pela *Microsoft Research*. Inicialmente ele foi desenvolvido com o regime de *software* proprietário (com código fechado), mas atualmente ele já adota o regime *open-source* e o seu código fonte pode ser encontrado em [3]. Ele pode ser utilizado para checar se fórmulas lógicas satisfazem uma ou mais teorias [1,2].

Z3 é considerada uma ferramenta de baixo nível. Sua utilização é feita normalmente por meio de *API*s (Ex: *Python* e *C*) e/ou em conjunto com outras ferramentas que precisam resolver fórmulas lógicas (Ex: *Boogie* [4] e *Dafny* [5]). Não existem *IDE*s independentes ou qualquer outra ferramenta voltada para o uso puramente do Z3[2].

A vantagem de utilizar provadores de teoremas ao invés de *model-checkers*, está no fato de que, para efetuar as análises e chegar à uma conclusão de que as propriedades são ou não satisfeitas, os *model-checkers* realizam os testes enumerando todas as possibilidades (estados), o que não acontece para provadores de teoremas. Além disso, os provadores de teorema facilitam a representação da passagem de tempo.

1.2 Provadores de Teoremas

Provadores de teoremas são utilizados para provar teoremas matemáticos. Dependendo da lógica escolhida, o problema de decidir a validade de uma conjectura pode variar do trivial at o impossível. Para os casos mais frequentes da lógica proposicional, os problemas são decifráveis mas são NP-completo [6]. Isto é, eles não podem ser resolvidos em tempo polinomial determinístico, mas a corretude de um dado resultado pode ser verificada. Como exemplo de um problema NP-completo, está o problema da soma dos subconjuntos. Um exemplo desse problema seria: dado um conjunto S de inteiros, determine se há algum conjunto não vazio de S cujos elementos somem 0. É fácil de verificar se uma resposta é correta, mas não se conhece uma solução significativamente mais rápida para resolver este problema do que testar todos os subconjuntos possíveis, até encontrar um que cumpra com a condição [8]. Para lógica de priemira ordem, os problemas são recursivamente enumeráveis. Isto é, dados recursos ilimitados, qualquer conjectua válida pode ser provada, mas conjecturas inválidas não podem ser sempre reconhecidas [6].

Como consequência desses problemas, pode ser observado que a aplicabilidade de provadores totalmente autmomáticos fica restringida, e a necessidade da intervenção humana na condução das provas nos chamados *provadores interativos de teoremas* ou *assistente de teoremas* se torna necessária [6].

1.3 Aplicações

Como Z3 é uma linguagem de baixo nível, normalmente as aplicações são feitas indiretamente, ou seja, existe alguma outra ferramenta que utiliza o Z3 como SMT, e essa que possui algumas aplicações.

Além disso, as aplicações diretas são mais complexas e normalmente são da área científica.

1.3.1 Aplicações Diretas

- **Sudoku solver** [13]
- **Thread Contracts for Safe Parallelism**
  Framework chamado de *Accord* que permite que aos programadores fazerem anotações para especificar explicitamente quais regiões da memória que uma *thread* pode ler ou escrever travando essas posições de memória. Para mais informações, ler [14]

1.3.2 Aplicações Indiretas

Dafny
Algumas aplicações de Dafny são:

- **IronClad (MSR, 30,000+ lines)** [11]
  IronClad App permite que o usuário transmita dados para uma máquina remota de forma segura e garantindo que todas as istruções que serão executadas nessa máquina aderem à uma especificação formal abstrata do comportamento do app. Além de eliminar vunerabilidades de implementação como *buffer overflows*, *parsing errors* ou *data leaks*, o IronCad diz ao usuário exatamente como o app vai se comportar por todo o tempo. Para mais informações, acessar o site da ferramenta em [10].
- **Specification and refinement of FreeRTOS** [11]
  Algumas especificações e refinamentos realizados no FreeRTOS foram realizadas utilizando Dafny. Para mais informações, ler [9].
- **ExpressOS (UIUC)** [11]
  Sistema Operacional para dispositivos móveis desenvolvido para garantir segurança ao utilizar os aplicativos. Para mais informações, ler [12].

2 Utilizando Z3

Como foi dito na sessão 1.1, não existe uma ferramente específica para trabalhar com Z3. Os códigos podem ser escritos em editores de texto comuns, e então serem interpretados utilizando um interpretador *online* fornecido pelo *rise4fun* [2], ou instalando o Z3 utilizando o código e o guia de instalação fornecido em [3] e executando o código escrito em Z3 via linha de comando.

3 Sintaxe

A sintaxe do Z3 é uma extensão da utilizada no *SMT-Lib 2.0 Standard*. Os códigos devem ser compostos por uma lista de comandos, e as fórmulas devem ser escritas em pré-ordem.

Na sessão 4 serão demonstrados alguns exemplos de códigos em Z3 aplicando os comandos listados na sessão 3.1.

3.1 Comandos Básicos

help
Mostra uma lista de comandos que podem ser utilizados.

echo
Mostra mensagens na tela.

Exemplo

```
1.  ; Escreve "starting Z3" na tela
2.  (echo "starting Z3...")
```

declare-const
Declara constantes ou *given types*.

Exemplo

```
; Declaração da variável "a"
(declare-const a Int)
```

declare-fun

Declara funções.

Exemplo

```
; Declaração da função "f" que recebe como argumentos uma variável do tipo
; "Int" (armazenada em "x!1") e uma do tipo "Bool" (armazenada em "x!2") e
; retorna uma do tipo "Int", que vai depender do resultado do comando ite
(define-fun f ((x!1 Int) (x!2 Bool)) Int
  (ite (and (= x!1 11) (= x!2 false)) 0 101)
)
```

assert

Adiciona uma fórmula na pilha interna do Z3. As fórmulas armazenadas na pilha interna são ditas como satisfeitas se houver uma resposta que satisfaça todas as fórmulas.

Exemplo1

```
; Verifica se a pode ser maior do que "10"
(assert (> a 10))
```

Exemplo2

```
; Verifica existe um resultado da função "f" passando como parâmetros "a"
; e "true" menor do que "100"
(assert (< (f a true) 100))
```

check-sat

Verifica se as fórmulas armazenadas na pilha podem ser satisfeitas (`sat`), não satisfeitas (`unsat`) ou se não foi possível determinar se poderam ou não serem satisfeitas (`unknown`).

Exemplo

```
(check-sat)
```

get-model

Mostra na tela um caso que satisfaz todas as fórmulas da pilha.

Exemplo

```
(get-model)
```

ite

Sigla de *if-then-else*.

Exemplo

```
; Retorna "21" se "x!1" for igual a "11" e "x!2" for igual a "false",
; e retorna "0" caso contrário
(ite (and (= x!1 11) (= x!2 false)) 21 0)
```

reset

Deleta todas as informaçoes armazenadas pelo Z3 até aquele ponto (declarações e afirmações).

Exemplo

```
(reset)
```

set-option

Usado para configurar o Z3.

Exemplo 1

```
; Escreve "success" ao executar uma linha de comando com sucesso
(set-option :print-success true)
```

Exemplo 2

```
1.  ; Mostra quais teorias não foram satisfeitas
2.  (set-option :produce-unsat-cores true)
```

Exemplo 3

```
1.  ; Ativa geração de modelos
2.  (set-option :produce-models true)
```

Exemplo 4

```
1.  ; Ativa geração de provas
2.  (set-option :produce-proofs true)
```

Exemplo 5

```
1.  ; Esta opção não pode ser selecionada depois de fazer alguma
2.  ; declaração ou teoria.
3.  (set-option :produce-proofs false)
```

get-value

Recupera valores de variáveis.

Exemplo

```
1.  ; Recupera os valores de "x" e "y"
2.  (get-value(x y))
```

get-unsat-core

Mostra na tela quais as *constraints* que juntas não poderam ser satisfeitas.

Exemplo

```
1.  (get-unsat-core(x y))
```

4 Exemplos

4.1 Exemplo 1

Exemplo simples para demonstração de declaração de variáveis e de funções, criação de *constraints*, e para demonstrar o comportamento dos comandos `echo`, `check-sat` e `get-model`.

Input

```
1.  (echo "starting Z3...")
2.  (declare-const a Int)
3.  (declare-fun f (Int Bool))
4.  (assert (> a 10))
5.  (assert (< (f a true) 20))
6.  (check-sat)
7.  (get-model)
```

Output

```
starting Z3...
sat
(model
  (define-fun a () Int
    11)
  (define-fun f ((x!1 Int) (x!2 Bool)) Int
    (ite (and (= x!1 11) (= x!2 true)) 0
      0))
)
```

4.2 Exemplo 2

Exemplo para demonstrar a definição de um escopo para a função e uma situação em que o `check-sat` chega à uma resposta `unsat`.

Input

```
1.  (echo "Funcao de teste")
2.  (declare-const a Int)
3.  (define-fun f ((x!1 Int) (x!2 Bool)) Int
4.    (ite (and (= x!1 11) (= x!2 false)) 0 101)
5.  )
6.  (assert (> a 10))
7.  (assert (< (f a true) 100))
8.  (check-sat)
```

Output

```
Funcao de teste
unsat
```

4.3 Exemplo 3

Exemplo no qual o `check-sat` encontra uma resposta e gera um modelo definindo um valor para a variável fazendo com que as *constraints* sejam satisfeitas. Observar que o `get-model` não pode vir antes do `check-sat`.

Input

```
1.  (echo "Funcao de teste")
2.  (declare-const a Int)
3.  (define-fun f ((x!1 Int) (x!2 Bool)) Int
4.    (ite (and (= x!1 11) (= x!2 false)) 0 101)
5.  )
6.  (assert (> a 10))
7.  (assert (< (f a false) 100))
8.  (check-sat)
9.  (get-model)
```

Output

```
Funcao de teste
sat
(model
  (define-fun a () Int
    11)
)
```

4.4 Exemplo 4

Exemplo utilizado para demonstrar o funcionamento do comando `get-value`. Observar que, para o Z3, as variáveis são interpretadas como funções cujo retorno é constante.

Input

```
1.  ; activate model generation
2.  (set-option :produce-models true)
3.
4.  (declare-fun x () Int)
5.  (declare-fun y1 () Int)
6.  (declare-fun y2 () Int)
7.  (declare-fun z () Int)
8.
9.  (assert (= x y1))
10. (assert (not (= y1 z)))
11. (assert (= x y2))
12. (assert (and (> y2 0) (< y2 5)))
13.
14. (check-sat)
15.
16. ; ask for the model value of some of the constants
17. (get-value (x z))
18.
19. (exit)
```

Output

```
sat
((x 1)
 (z 2))
```

4.5 Exemplo 5

Exemplo utilizando números reais. Observe que, ao utilizar o `get-model`, o Z3 representa os números reais como sendo uma divisão. Isto é uma indicação de que, ao realizar os cálculos durante o processamento das *constraints*, os números reais são transformados em frações, o que acontece provavelmente por motivos de otimização.

Input

```
1.  (declare-const a Real)
2.  (assert (> a (/ 3.75 1.7)))
4.  (check-sat)
5.  (get-model)
```

Output

```
sat
(model
  (define-fun a () Real
    (/ 109.0 34.0))
)
```

4.6 Exemplo 6

Exemplo para demonstrar o funcionamento do `get-unsat-core` e como nomear as funções utilizando `:named`.

Input

```
1.  ; activate unsat core computation
2.  (set-option :produce-unsat-cores true)
3.
4.  (declare-fun x () Int)
5.  (declare-fun y1 () Int)
6.  (declare-fun y2 () Int)
7.  (declare-fun z () Int)
8.
9.  (define-fun A1 () Bool (= x y1))
10. (define-fun A2 () Bool (not (< z 0)))
11. (define-fun A3 () Bool (= y1 z))
12. (define-fun B () Bool (and (= x y2) (not (= y2 z))))
13.
14. ; use annotation :named to give a name to toplevel formulas.  The
15. ; unsatisfiable core will be a subset of the named formulas.  If a toplevel
16. ; formula doesn't have a name, it will not be part of the unsat core
17. (assert (!  A1 :named First))
18. (assert (!  A2 :named Second))
19. (assert (!  A3 :named Third))
20. (assert (!  B :named Fourth))
21.
22. (check-sat)
23. (get-unsat-core)
24.
25. (exit)
```

Output

```
unsat
(Fourth First Third)
```

5 Acrônimos

1. API: Application Programming Interface
2. FreeRTOS: Free Real Time Operating System
3. MSR: Microsoft Research
4. OS: Operating System
5. SMT: Satisfiability Modulo Theories
6. UIUC: University of Illinois Urbana-Champaign

6 Links

1. Microsoft Research Z3 website
2. rise4fun Z3
3. Z3 Source-code
4. Boogie
5. Dafny
6. Provadores de teoremas - PUC
7. SMT-LIB
8. Problema NP-Completo
9. Program Verification of FreeRTOS using Microsoft Dafny
10. IronClad
11. Aplicações de Dafny
12. ExpressOS
13. Sudoku Solver
14. Thread Contracts for Safe Parallelism]