Universidade Federal do Amazonas
Faculdade de Tecnologia
PROGRAMA DE PÓS-GRADUAÇÃO EM ENGENHARIA ELÉTRICA

Teorias do Módulo da Satisfatibilidade (*Satisfiability Modulo Theories* - SMT)

Lucas C. Cordeiro
lucascordeiro@ufam.edu.br

Teorias do Módulo da Satisfatibilidade

SMT decide a **satisfatibilidade** de fórmulas de primeira-ordem usando a combinação de diferentes **teorias de fundamentação (background)**

Theory	**Example**
Equality	$x_1=x_2 \wedge \neg (x_2=x_3) \Rightarrow \neg(x_1=x_3)$
Bit-vectors	(b >> i) & 1 = 1
Linear arithmetic	$(4y_1 + 3y_2 \geq 4) \vee (y_2 - 3y_3 \leq 3)$
Arrays	$(j = k \wedge a[k]=2) \Rightarrow a[j]=2$
Combined theories	$(j \leq k \wedge a[j]=2) \Rightarrow a[i] < 3$

Teorias do Módulo da Satisfatibilidade

- Dado
  - uma teoria $\sum$-theory T
  - uma fórmula φ livre de quantificadores

  φ é satisfatível em *T* se e somente se $T \cup \{\varphi\}$ é satisfatível, i.e., existe uma *estrutura* que *satisfaz* as *fórmulas* e *sentenças* de T
- Dado
  - um conjunto $\Gamma \cup \{\varphi\}$ de fórmulas de primeira-ordem sobre T

  φ é uma consequência da *T* de Γ ($\Gamma \models_T \varphi$) se e somente *cada modelo de* $T \cup \Gamma$ é também um modelo de φ
- Verificação $\Gamma \models_T \varphi$ pode ser reduzido usualmente através da verificação de satisfatibilidade em T de $\Gamma \cup \{\neg\varphi\}$

Teorias do Módulo da Satisfatibilidade

- Considere **a** como um vetor, **b**, **c** e **d** como vetores de bit sinalizados de comprimento 16, 32 e 32 respectivamente, e considere **g** como uma função unária

$$g\,(select\ \ (store\ \ (a\,,c\,,12\,)),\ SignExt\ \ \ (b\,,16\,)+\,3\,)$$
$$\neq\ g\,(SignExt\ \ \ (b\,,16\,)-\,c\,+\,4\,)\wedge\ SignExt\ \ \ (b\,,16\,)=\,c\,-\,3\,\wedge\,c\,+1\,=\,d\,-\,4$$

⇩ **b'** estende **b** para o vetor de bit equivalente de tamanho 32

$$step\,1:g\left(select\left(store\left(a,c,12\right),b'+3\right)\right)\neq g\left(b'-c+4\right)\wedge b'=c-3\wedge c+1=d-4$$

⇩ substitui b' por c–3 na desigualdade

$$step\,2:g\left(select\left(store\left(a,c,12\right),c-3+3\right)\right)\neq g\left(c-3-c+4\right)\wedge c-3=c-3\wedge c+1=d-4$$

⇩ usando artefatos de aritmética de vetor de bit

$$step\,3:g\left(select\left(store\left(a,c,12\right),c\right)\right)\neq g\left(1\right)\wedge c-3=c-3\wedge c+1=d-4$$

Teorias do Módulo da Satisfatibilidade

$$step\ 3 : g(select(store(a, c, 12), c)) \neq g(1) \wedge c - 3 = c - 3 \wedge c + 1 = d - 4$$

⇩ aplicando a teoria de vetores

$$step\ 4 : g(12) \neq g(1) \wedge c - 3 = c - 3 \wedge c + 1 = d - 4$$

⇩ A função g implica que para todo x e y,
se x = y, então g (x) = g (y) (*congruence rule*)

$$step\ 5 : SAT\ (c = 5, d = 10)$$

- Solucionadores SMT também aplicam:
  - normas-padrão de redução algébricas $\boxed{r \wedge false \mapsto false}$
  - simplificação contextual $\boxed{a = 7 \wedge p(a) \mapsto a = 7 \wedge p(7)}$

Solucionador SMT Z3

- O Z3 é um solucionador das teorias do módulo da satisfatibilidade que é estado da arte
  - desenvolvido e mantido pela *Microsoft Research* (Redmond)
- O Z3 é utilizado para verificar a satisfatibilidade de fórmulas em lógica de primeira ordem
  - integra vários procedimentos de decisão
    - ▷ aritmética inteira e real linear, vetores de bit de tamanho fixo, funções não interpretadas, arrays e quantificadores
- É uma ferramenta usada para análise de programas, verificação e geração de casos de teste na Microsoft
  - normalmente integrada a outras ferramentas através de sua API escrita em C/C++ ou através da SMT-lib

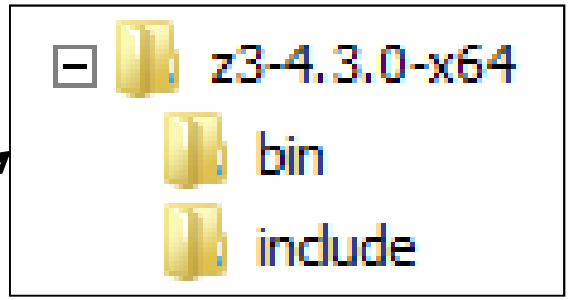
Exemplo: API do C

- O solucionador deve ser rápido em instâncias que sejam satisfeitas

```
for (n = 2; n <= 5; n++) {
    printf("n = %d\n", n);
    ctx = Z3_mk_context(cfg);

    bool_type   = Z3_mk_bool_type(ctx);
    array_type  = Z3_mk_array_type(ctx, bool_type, bool_type);

    /* create arrays */
    for (i = 0; i < n; i++) {
        Z3_symbol s = Z3_mk_int_symbol(ctx, i);
        a[i]        = Z3_mk_const(ctx, s, array_type);
    }

    /* assert distinct(a[0], ..., a[n]) */
    d = Z3_mk_distinct(ctx, n, a);
    printf("%s\n", Z3_ast_to_string(ctx, d));
    Z3_assert_cnstr(ctx, d);

    /* context is satisfiable if n < 5 */
    if (Z3_check(ctx) == l_false)
        printf("unsatisfiable, n: %d\n", n);

    Z3_del_context(ctx);
}
```

Configurando o Solucionador SMT Z3

- Download e configuração
  - Link para baixar: http://z3.codeplex.com/releases
  - Descompactar e executar:
    - ▷ Descompactar e criar a pasta



    - ▷ Executar Z3 através do CMD do Windows

```
C:\Temp\z3-4.3.0-x64\bin>z3 /h
```

Configurando o Solucionador SMT Z3

- Configuração no Linux
  - Executando aplicações Windows
    - ▹ Baixar *Wine*: ***sudo apt-get install wine1.4-i386***
    - ▹ Executando: ***wine z3.exe /h***
  - Como construir o Z3 para o Linux
    - ▹ Baixar o código fonte: git clone https://git01.codeplex.com/z3
    - ▹ Configurando:

      autoconf

      ./configure

      python scripts/mk_make.py

      cd build

      Make

      chmod 751 z3
    - ▹ Executando: ***~/repositories/z3/build/z3 -version***

Como Funciona o Solucionador SMT Z3

- O Z3 executa um *script* que é uma sequência de comandos
- Os comandos em Z3 são baseados no padrão SMT-LIB 2.0
  - http://smt-lib.org/
  - http://www.grammatech.com/resource/smt/SMTLIBTutorial.pdf
  - http://smtlib.cs.uiowa.edu/papers/smt-lib-reference-v2.0-r10.12.21.pdf
- O Z3 mantém uma pilha das declarações e fórmulas fornecidas pelo usuário
  - O comando ***declare-const*** declara uma constante de um dado tipo
  - O comando ***declare-fun*** declara uma função
  - O comando ***assert*** adiciona uma fórmula dentro da pilha interna do Z3

Entendendo o Solucionador SMT Z3

- Uma fórmula *P* na pilha do Z3 é **satisfatível** se existe alguma atribuição de valores apropriados para os símbolos das suas funções sob qual *P* avalia para **verdadeiro**
  - Uma fórmula *P* é **válida** se *P* sempre avalia para verdadeiro para qualquer atribuição de valores apropriados
- O comando ***check-sat*** retorna ***sat*** ou ***unsat*** se a fórmula é satisfatível ou não satisfatível, respectivamente.
  - Caso não seja ***sat*** ou ***unsat***, retorna ***unknow***
- O comando ***get-model*** pode ser usado para capturar informações da pilha Z3 após a interpretação das fórmulas
- O link http://www.rise4fun.com/Z3 é uma interface on-line que permite executar interativamente comandos SMT Z3

Executando o Solucionador SMT Z3

Exercício 1: Liste as linhas de comando do solucionador Z3

- *C:\z3-4.3.0-x64\bin> z3 /h*

```
Z3 [version 4.3.0 - 64 bit]. (C) Copyright 2006 Microsoft Corp.
Usage: z3 [options] [/file:]file

Input format:
  /smt          use parser for SMT input format.
  /smt2         use parser for SMT 2 input format.
  /dl           use parser for Datalog input format.
  /dimacs       use parser for DIMACS input format.
  /log          use parser for Z3 log input format.
  /in           read formula from standard input.

Miscellaneous:
  /h, /?        prints this message.
  /version      prints version number of Z3.
  /v:level      be verbose, where <level> is the verbosity level.
  /nw           disable warning messages.
  /ini:file     configuration file.
  /ini?         display all available INI file parameters.
  --            all remaining arguments are assumed to be part of the input file n
ame. This option allows Z3 to read files with strange names such as: -foo.smt2.

Resources:
  /T:timeout    set the timeout (in seconds).
  /t:timeout    set the soft timeout (in seconds). It only kills the current query
.
  /memory:Megabytes  set a limit for virtual memory consumption.

Output:
  /st           display statistics.

Search heuristics:
  /rs:num       random seed.
```

Executando o Solucionador SMT Z3

Exercício 2: Como executar um *script* SMT no Z3?

- Crie e edite um arquivo SMT
  - ***notepad scriptZ3.smt2***
- Adicione os comandos dentro do *script*

```
(echo "starting Z3...")
(set-logic QF_UF)
(declare-fun p () Bool)
(assert (and p (not p)))
(check-sat)
(exit)
```

Declara uma variável booleana *p* e pergunta se ($p \wedge \neg p$) é satisfatível

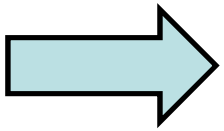
- Execute o *scriptZ3.smt2*
  - ***z3 /smt2 scriptZ3.smt2***

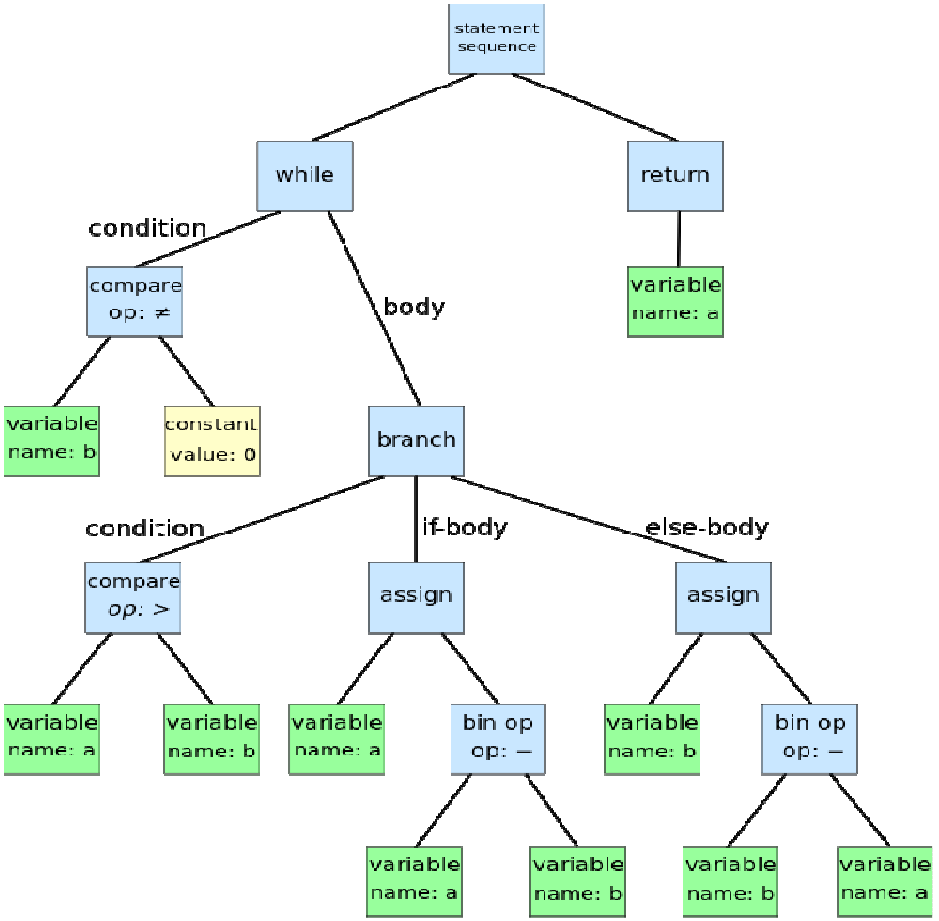
```
C:\Temp\z3-4.3.2-x64-win\bin>z3 /smt2 scriptZ3.smt2
starting Z3...
unsat
```

Árvore de Análise Sintática

- **AST (*Abstract Syntax Tree*):** é uma estrutura de dados em árvore que permite criar uma representação compacta e fácil de trabalhar da estrutura de programas ou fórmulas

```
while b ≠ 0
    if a > b
        a := a − b
    else
        b := b − a
return a
```



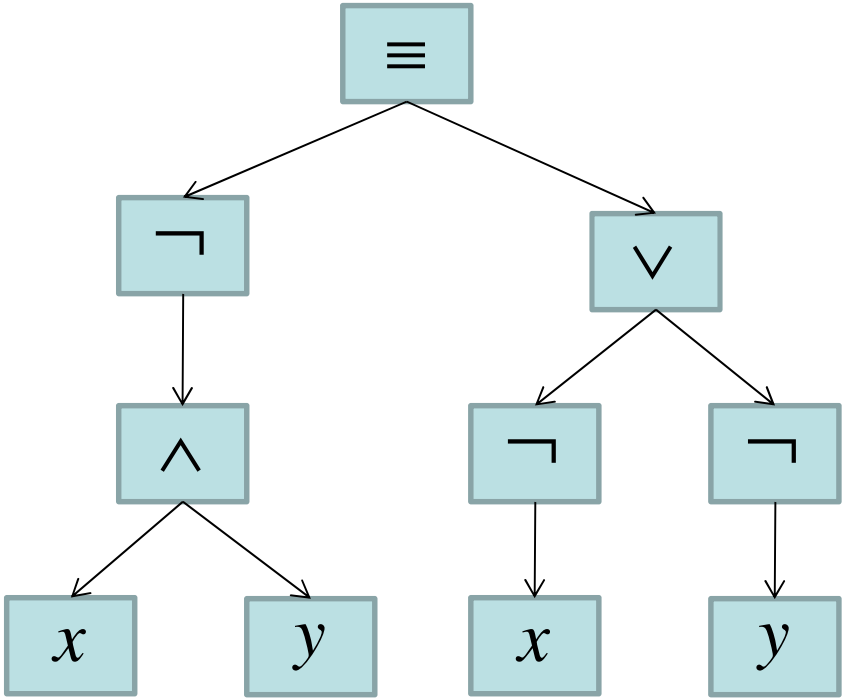


Representando Fórmulas no Z3

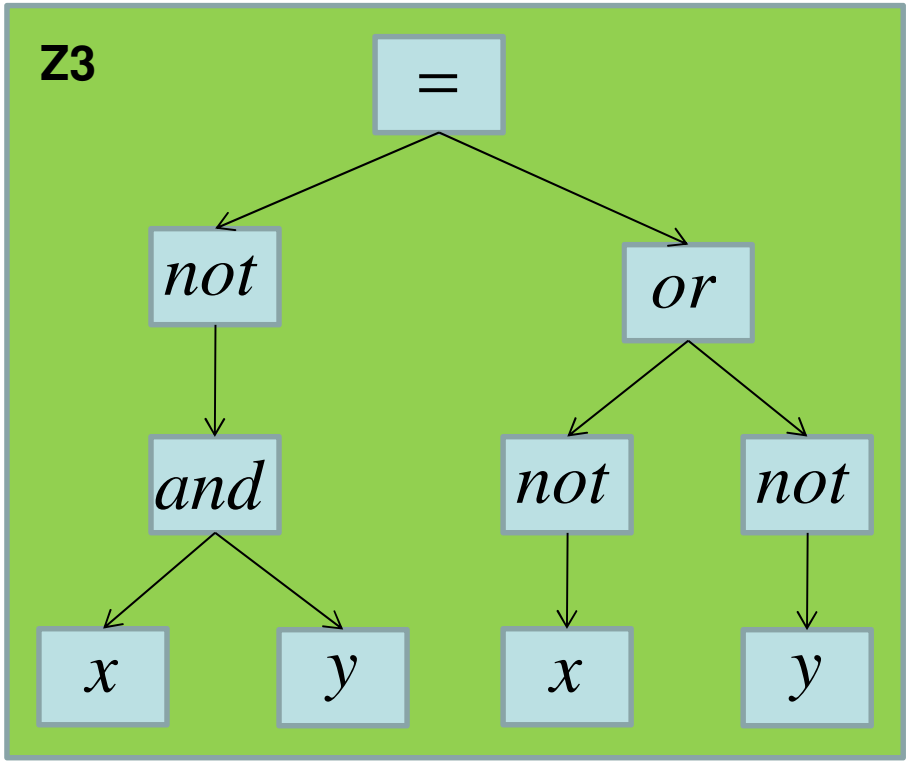
Exercício 3: Utilizando Z3 verifique se a fórmula abaixo é satisfatível ou não

$$\neg(x \wedge y) \equiv (\neg x \vee \neg y)$$

1°) Montar a AST para a fórmula



$\equiv (\neg (\wedge \ x \ y) (\vee (\neg \ x) (\neg \ y))$



$= (not \ (and \ x \ y) (or \ (not \ x) (not \ y))$

Executando Fórmulas no Z3

Exercício 3: Utilizando Z3 verifique se a fórmula abaixo é satisfatível ou não

$$\neg(x \wedge y) \equiv (\neg x \vee \neg y)$$

2°) Escrever o *script* SMT

$= (not\ (and\ \ x\ \ y)\ (or\ (not\ \ x)\ (not\ \ y))$

```
(declare-fun x () Bool)
(declare-fun y () Bool)
(assert ( = (not(and x y)) (or (not x) (not y))))
(check-sat)
(get-value (x y))
```

3°) Executar o *script* SMT

```
z3 /smt2 scriptZ3.smt2
```

```
C:\Temp\z3-4.3.2-x64-win\bin>z3 /smt2 scriptZ3.smt2
sat
((x false)
 (y false))
```

Executando Fórmulas no Z3

Exercício 3: Mostre que as duas expressões **if-then-else** abaixo são equivalentes:

```
!(a||b)?h:!(a==b)?f:g        !(!a||!b)?g:(!a&&!b)?h:f
```

Escrever o *script* SMT

```
(declare-fun a() Bool)
(declare-fun b() Bool)
(declare-fun f() Bool)
(declare-fun g() Bool)
(declare-fun h() Bool)
(declare-fun expr1() Bool)
(declare-fun expr2() Bool)
(assert (= expr1 (ite (not (or a b)) h (ite  (not (= a b))
  f g ))))
(assert (= expr2 (ite (not (or (not a) (not b))) g (ite
  (and (not a) (not b)) h f))))
(assert (not (= expr1 expr2)))
(check-sat)
```

Exercícios com o Solucionador Z3

Exercício 4: Utilizando Z3 determine se as seguintes fórmulas são satisfeitas

a) $\neg(x \vee y) \equiv (\neg x \wedge \neg y)$

b) $(x \wedge y) \equiv \neg(\neg x \vee \neg y)$

Equações com Aritmética Linear Inteira

Exercício 5: Resolvendo equações com Z3

1°) Escrever o *script* SMT

```
(set-logic QF_LIA)
(declare-fun x() Int)
(declare-fun y() Int)
(assert (= (+ x y) 10))
(assert (= (+ x (* 2 y)) 20))
(check-sat)
(get-model) ; captura os resultados da pilha Z3
```

x+y=10

x+2*y=20

Comentário

2°) Executar o *script* SMT

```
z3 /smt2 scriptZ3.smt2
```

```
C:\Temp\z3-4.3.2-x64-win\bin>z3 /smt2 scriptZ3.smt2
sat
(model
  (define-fun y () Int
    10)
  (define-fun x () Int
    0)
)
```

Exercícios com o Solucionador Z3

Exercício 6: Resolva as equações com Z3

a) $3x + 2y = 36$

b) $5x + 4y = 64$

Usando Quantificadores no Z3

Exercício 7: Verifique a seguinte fórmula usando Z3

$$\forall x : \mathrm{I}nt, x \leq a \Rightarrow x < b$$

1°) Escrever o *script* SMT

```
(declare-const a Int)
(declare-const b Int)
(assert (forall ((x Int)) (=> (<= x a) (< x b))))
(check-sat-using (then qe smt))
(get-model) ; captura os resultados da pilha Z3
```

quantifier elimination

2°) Executar o *script* SMT

```
z3 /smt2 scriptZ3.smt2
```

```
C:\Temp\z3-4.3.2-x64-win\bin>z3 /smt2 scriptZ3.smt2
sat
(model
  (define-fun b () Int
    1)
  (define-fun a () Int
    0)
)
```

Convertendo Código C para SMT-LIB

Exercício 8: Converta para Z3 a seguinte estrutura

```
int f(int x, int y) {
  if(x==11&&!y)
    return 21
  else
     return 0;
}
```

Script SMT

```
(declare-const a Int)
(define-fun f((x!1 Int) (x!2 Bool)) Int
   (ite (and (= x!1 11) (= x!2 false)) 21 0)
 )
(assert (= a 11))
(assert(< (f a true) 100))
(check-sat)
```

Convertendo Código C para SMT

Exercício 9: Converta o seguinte código C para Z3

```
int main() {
  int a[2], i, x;
  if (x==0)
    a[i]=0;
  else
    a[i+2]=1;
  assert(a[i+1]==1);
}
```

$$C := \begin{bmatrix} g_1 := (x_1 = 0) \\ \wedge\, a_1 := store(a_0, i_0, 0) \\ \wedge\, a_2 := a_0 \\ \wedge\, a_3 := store(a_2, 2+i_0, 1) \\ \wedge\, a_4 := ite(g_1, a_1, a_3) \end{bmatrix}$$

$$P := \begin{bmatrix} i_0 \geq 0 \wedge i_0 < 2 \\ \wedge\, 2+i_0 \geq 0 \wedge 2+i_0 < 2 \\ \wedge\, 1+i_0 \geq 0 \wedge 1+i_0 < 2 \\ \wedge\, select(a_4, i_0+1) = 1 \end{bmatrix}$$

Exemplo de código usando a API do Z3

```
Z3_context ctx;
Z3_ast g1, x1, a0, a1, a2, a3, a4, i0, C[5], P[4],
       zero, one, two, constraints, properties, l0, l1, l2, l3;
Z3_sort bool_sort, int_sort, array_sort;
ctx = mk_context();
bool_sort   = Z3_mk_bool_sort(ctx);
int_sort    = Z3_mk_int_sort(ctx);
array_sort  = Z3_mk_array_sort(ctx, int_sort, int_sort);
zero = mk_int(ctx, 0);
...
```

Convertendo Código C para SMT

Conjunto de fórmulas no formato da SMT-lib

```
constraints: (and (= g1 (= x1 0))
     (= a1 (store a0 i0 0))
     (= a2 a0)
     (= a3 (store a2 (+ i0 2) 1))
     (= a4 (if g1 a1 a3)))
properties: (and (and (>= i0 0) (< i0 2))
     (and (>= (+ i0 2) 0) (< (+ i0 2) 2))
     (and (>= (+ i0 1) 0) (< (+ i0 1) 2))
     (= (select a4 (+ i0 1)) 1))
formula: (and (and (= g1 (= x1 0))
           (= a1 (store a0 i0 0))
           (= a2 a0)
           (= a3 (store a2 (+ i0 2) 1))
           (= a4 (if g1 a1 a3)))
     (not (and (and (>= i0 0) (< i0 2))
               (and (>= (+ i0 2) 0) (< (+ i0 2) 2))
               (and (>= (+ i0 1) 0) (< (+ i0 1) 2))
               (= (select a4 (+ i0 1)) 1))))
```

Convertendo Código C para SMT

Resultado da checagem da fórmula

```
checking formula...
sat
a3 = (define as-array[k!1] (Array Int Int))
a2 = (define as-array[k!0] (Array Int Int))
a4 = (define as-array[k!2] (Array Int Int))
a1 = (define as-array[k!2] (Array Int Int))
l2 = (define true Bool)
i0 = -1:int
a0 = (define as-array[k!0] (Array Int Int))
g1 = (define true Bool)
l0 = (define false Bool)
x1 = 0:int
l1 = (define true Bool)
l3 = (define false Bool)
```

Convertendo Código C para SMT-LIB

Trabalho 1: Converta o seguinte código C para Z3

```
float c; int n=4;
while(n>0){
    c = 10/n;
    n--;
}
```

Pesquise os links abaixo para verificar como implementar o código acima em um *script* SMT.

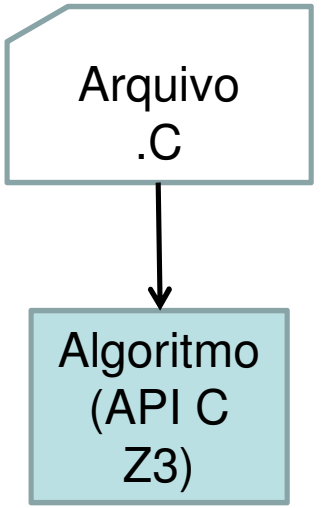
http://rise4fun.com/z3/tutorial

http://www.grammatech.com/resource/smt/SMTLIBTutorial.pdf

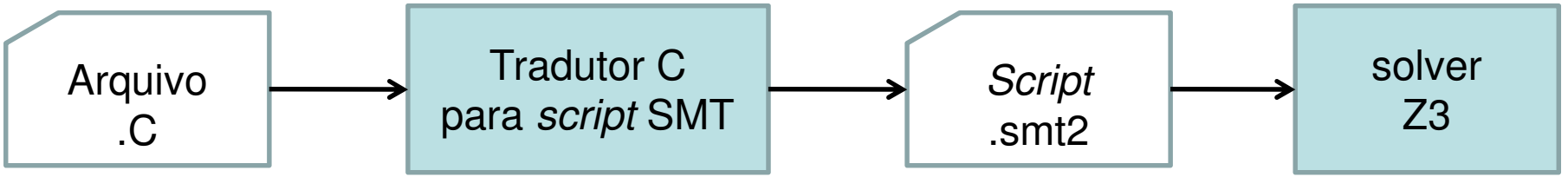
Convertendo Código C para C API do Z3

Trabalho 2: Desenvolver um algoritmo em C, utilizando API C do Z3, para verificar o código abaixo.

```
float c; int n=4;
while(n>0){
    c = 10/n;
    n--;
}
```



Outra forma de resolver é desenvolver um ***tradutor*** de código C para script SMT



Alguns Links Úteis do Z3

- Documentação / Tutoriais Z3

  http://research.microsoft.com/en-us/um/redmond/projects/z3/old/documentation.html#slides

- Command Line Options

  http://research.microsoft.com/en-us/um/redmond/projects/z3/old/cmdline.html

- C API

  http://research.microsoft.com/en-us/um/redmond/projects/z3/old/group__capi.html

- Theory plugin examples

  http://research.microsoft.com/en-us/um/redmond/projects/z3/old/group__theory__plugin__ex.html