SMT and Its Application in **Software Verification**

Yu-Fang Chen
IIS, Academia Sinica

Based on the slides of Barrett, Sanjit, Kroening , Rummer, Sinha, Jhala, and Majumdar

Assertion in C

```
int main(){
    int x;
    scanf("%d", &x);
    assert(x > 10);  }
```

- Useful tool for debugging.
  - Can be used to describe pre- and post-conditions of a function.
  - A program terminates immediately, if it reaches a **violated** assertion.

```
[yfc@FM3 ~]$ gcc test.c
[yfc@FM3 ~]$ ./a.out
10
a.out: test.c:9: main: Assertion `x > 10' failed.
Aborted
```

Assertion in C

```
int main(){
    int x;
    scanf("%d", &x);
    while(x<10){
        x++;
    }
    assert(x > 0);
}
```

- Will this assertion be violated?

Assertion in C

```
int main(){
    int x;
    scanf("%d", &x);
    while(x<10){
        x--;
    }
    assert(x > 0);
}
```

- Will this assertion be violated?

Assertion in C

```
int main(){
    int x;
    scanf("%d", &x);
    while(x<4324358){
        x--;
    }
    assert(x > 4324358);  }
```

- Will this assertion be violated?

Assertion in C

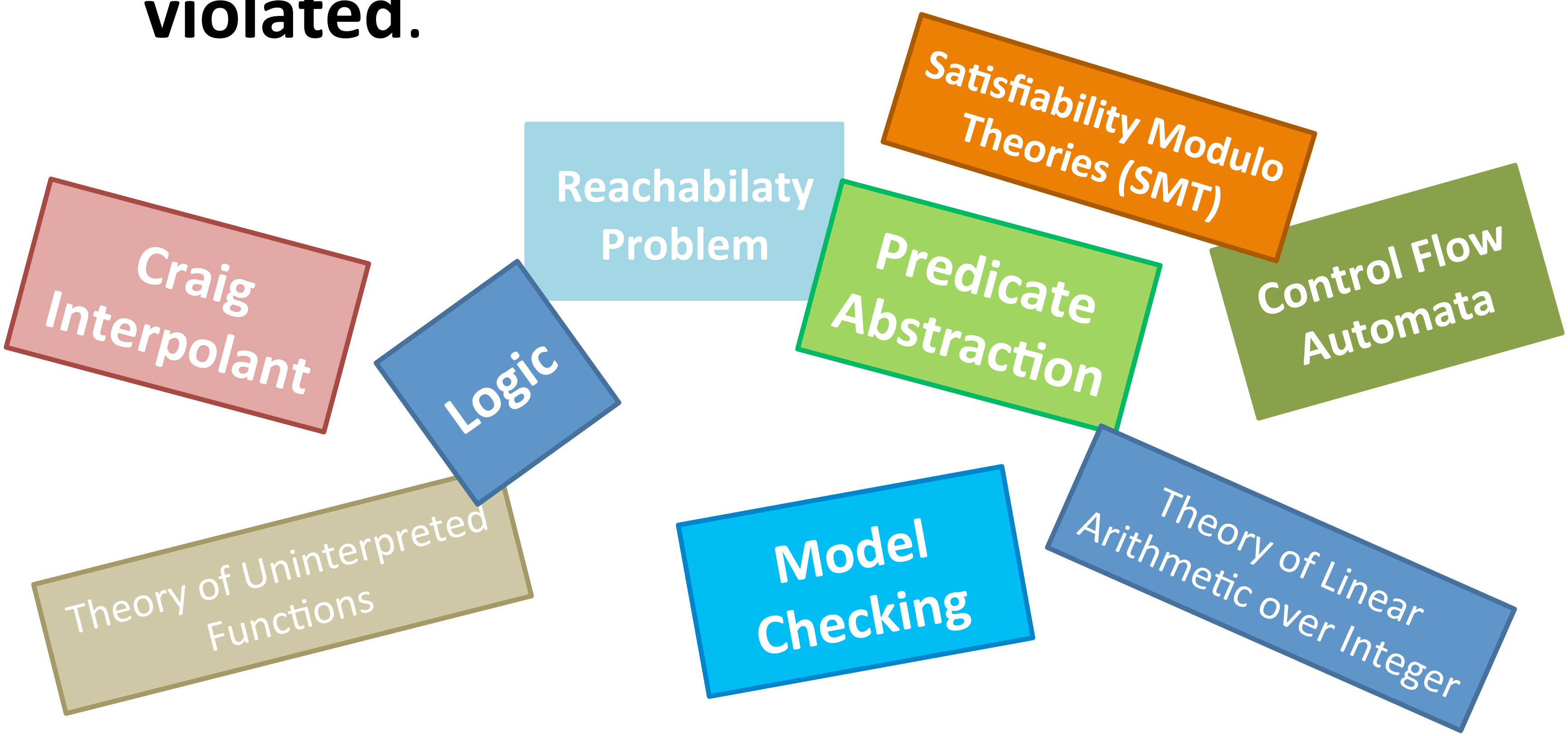
- One more example:

```
void A(bool h, bool g){
  h := !g;
  g=B(g,h);
  g=B(g,h);
  assert(g);
}
```

```
void B(bool a1,bool a2){
  if (a1)
    return B(a2,a1);
  else
    return a2;
}
```

The Problem We are Going to Solve

- Given a program with assertion, we want to **automatically detect if the assertion may be violated**.



Part I: Logic and Program Verification

First-Order Logic: A Quick Review

- Logical Symbols
  - Propositional connectives: $\vee$, $\wedge$, $\neg$, $\rightarrow$, $\leftrightarrow$
  - Variables: v1, v2, . . .
  - Quantifiers: $\forall$, $\exists$
- Non-logical Symbols
  - Functions: +, -, *, suc, …
  - Predicates: $\leq$, = , …
  - Constant symbols: 0, 1, null, …
- Example
  - $3*v2 + 5*v1 \leq 54$

Why This is Relevant to Software Verification?

For example:

- Given an integer program without loop and function call.
- The **assertion checking** problem can be reduced to $\boxed{\textbf{satisfibility}\text{ problem}}$ of a trace formula*

```
int  main(){
    int x; scanf("%d", &x);
    if(x<10)  x= x -1;
    assert(x != 9);
}
```

$$(x_0 < 10 \land x_1 = x_0 - 1 \land x_1 = 9) \land (x_0 \geq 10 \land x_0 = 9)$$

The assertion may be violated **iff** The first order formula is satisfible

Note *: a FOL formula under theory of linear integer arithmetic.

First order logic Theories

- A first order theory T consists of
  - Variables
  - Logical symbols: $\wedge$ $\vee$ $\neg$ $\forall$ $\exists$ `(’ `)’
  - **Signature Σ: Constants, predicate and function symbols**
  - **The meanings of the signatures.**

Examples

- Theory T:
  - Σ = {0,1, '+', '='}
    - '0','1' are constant symbols
    - '+' is a binary function symbol
    - '=' is a binary predicate symbol
- An example of a T-formula:

$$\exists x.\ x + 0 = 1$$

Is it T-valid?

Structures

- The most common way of specifying the meaning of symbols is to specify a **structure**

- Recall that F = $\exists$x. x + 0 = 1
- Consider the structure S:
  - Domain: $\mathcal{N}_0$
  - Interpretation of the non-logical symbols :
    - ‘0’ and ‘1’ are mapped to 0 and 1 in $\mathcal{N}_0$
    - ‘=’ $\mapsto$ = (equality)
    - ‘+’ $\mapsto$ * (multiplication)
- Now, is F valid under S ?

Short Summary

- A theory defines
  - the signature Σ (the set of non-logical symbols) and
  - the interpretations that we can give them.

Theories through axioms

- The number of sentences that are necessary for defining a theory may be large or infinite.
- Instead, it is common to define a theory through a set of axioms.
- The theory is defined by these axioms and everything that can be inferred from them by a sound inference system.

Example 1

- Let Σ = {‘=’}
  - An example is $F=((x = y) \wedge \neg (y = z)) \rightarrow \neg(x = z)$
- We would now like to define a theory T that will limit the interpretation of ‘=‘ to equality.
- We will do so with the equality axioms:
  - $\forall x.\ x = x$ (reflexivity)
  - $\forall x,y.\ x = y \rightarrow y = x$ (symmetry)
  - $\forall x,y,z.\ x = y \wedge y = z \rightarrow x = z$ (transitivity)
- Every assignment that satisfies these axioms also satisfies F above.
- Hence F is T-valid.

Example 2

- Let Σ = {'<'}
- Consider the formula F = $\forall x\ \exists y.\ y < x$
- Consider the theory T with axioms:
  - $\forall x,y,z.\ x < y \land y < z \rightarrow x < z$ (transitivity)
  - $\forall x,y.\ x < y \rightarrow \neg(y < x)$ (anti-symmetry)

Some Useful Theories in Software Verification

- **Equality (with uninterpreted functions)**
- **Linear arithmetic (over $\mathbb{Q}$ or $\mathbb{Z}$)**
  - Peano Arithmetic, Presburgh Arithmetic
- Difference logic (over $\mathbb{Q}$ or $\mathbb{Z}$)
- Finite-precision bit-vectors
  - integer or floating-point
- Arrays
- Misc.: strings, lists, sets, …

Theory of Equality and Uninterpreted Functions (EUF)

- Signature:
  - Constants and Function symbols: f, g, etc. In principle, all possible symbols but "=" and those used for variables.
  - Predicates symbol: "="
- equality axioms:
  - $\forall x.\ x = x$ (reflexivity)
  - $\forall x,y.\ x = y \rightarrow y = x$ (symmetry)
  - $\forall x,y,z.\ x = y \wedge y = z \rightarrow x = z$ (transitivity)
- In addition, we need *congruence*: the function symbols map identical arguments to identical values, i.e., $x = y \Rightarrow f(x) = f(y)$

Example EUF Formula

$$(x = y) \wedge (y = z) \wedge (f(x) = f(z))$$

Transitivity:

$$(x = y) \wedge (y = z) \Rightarrow (x = z)$$

Congruence:

$$(x = z) \Rightarrow (f(x) = f(z))$$

Equivalence Checking of Program Fragments

```
int fun1(int y) {
   int x, z;
   z = y;
   y = x;
   x = z;

  return sq(x);
}

int fun2(int y) {
    return sq(y);
}
```

The formula is satisfiable iff
the programs are non-equivalent

$$z0 = y0 \wedge y1 = x0 \wedge x1 = z0 \wedge ret1 = sq(x1)$$
$$\wedge$$
$$ret2 = sq(y0)$$
$$\wedge$$
$$ret1 = ret2$$

A Small Practice:

```
int f(int y) {
    int x, z;
    x = myFunc(y);
    x = myFunc(x);
    z = myFunc(myFunc(y));

   assert (x==z);
}
```

Write a formula F such that

Formula F is satisfible $\leftrightarrow$
the assertion can be violated

Solution:

First Order Peano Arithmetic

- $\Sigma = \{\underbrace{0,1}_{\text{constant}},\underbrace{\text{`}+\text{'}, \text{`}\times\text{'}}_{\text{function}}, \underbrace{\text{`}=\text{'}}_{\text{predicate}}\}$
- Domain: Natural numbers

Validity is

$\boxed{\textit{Undecidable!}}$

- Axioms ("semantics"):
  1. $\forall x : \neg(0 = x + 1)$
  2. $\forall x : \forall y : \neg(x=y) \rightarrow \neg(x + 1=y + 1)$
  3. Induction
  4. $\forall x : x + 0 = x$
  5. $\forall x : \forall y : (x + y) + 1 = x + (y + 1)$
  6. $\forall x : x \times 0 = 0$
  7. $\forall x \forall y : x \times (y + 1) = x \times y + x$

$+$ (axioms 4–5); $\times$ (axioms 6–7)

These axioms define the semantics of '+'

First Order Presburger Arithmetic

- $\Sigma = \{\overbrace{0,1}^{\text{constant}}, `+\text{'}, \overbrace{\quad}^{\text{function}}, \overbrace{`=\text{'}}^{\text{predicate}}\}$
- Domain: Natural numbers

Validity is

$\boxed{\textit{Decidable!}}$

- Axioms ("semantics"):
  1. $\forall x : \neg(0 = x + 1)$
  2. $\forall x : \forall y : \neg(x = y) \rightarrow \neg(x + 1 = y + 1)$
  3. Induction

$$+\left\{\begin{array}{l} 4.\quad \forall x : x + 0 = x \\ 5.\quad \forall x : \forall y : (x + y) + 1 = x + (y + 1) \end{array}\right\}$$

These axioms define the semantics of '+'

$\boxed{\text{Note that } 3 \times v2 + 5 \times v1 \leq 54 \text{ is a Presburger Formula}}$

Examples in Software Verification

- Array Bound Checking:

```
void f() {
   int x[10];
   x[0]=1;
   for(int i=1; i<10; i++){
      assert(0<i<10);
      x[i]=x[i-1]+5;
      ……
   }
}
```

$i0=1 \wedge i1=i0+1 \wedge i2=i1+1 \wedge \ldots \wedge i8=i7+1 \wedge$
$\neg(0<i0<10 \wedge 0<i1<10 \wedge \ldots \wedge 0<i8<10)$
is satisfible iff
the assertion may be violated

x>y can be translated to $\exists u \in N_0$: $x= y+u$

Theory of Arrays

- Two interpreted functions: select and store
  - select(A,i) Read from array A at index i
  - store(A,i,d) Write d to array A at index i
- Two main axioms:
  - select(store(A,i,d), i) = d
  - select(store(A,i,d), j) = select(A,j) for $\neg$(i = j)
- Extentionality axiom:
  - $\forall$i. select(A,i) = select(B,i) $\rightarrow$ (A = B)

Combining Theories

- **Satisfiability Modulo Theories** (**SMT**) **problem** is a decision problem for logical formulae with respect to combinations of different first order theories.
- For example: Uninterpreted Function + Linear Integer Arithmetic

$$1 \leq x \land x \leq 2 \;\land\; f(x) \neq f(1) \land f(x) \neq f(2)$$

Linear Integer Arithmetic (LIA)	Uninterpreted Functions(UF)

- How to Combine Theory Solvers?

  A Classical Algorithm: The Nelson-Oppen Method.

What you have learned so far?

- Assertions in C
- First Order Theories related to Verification
  - Equality and Uninterpreted Functions

    $f(a,b)=a \wedge f(f(a, b), b)=a$
  - Linear Integer Arithmetic

    $x+5>y-7 \wedge 3y < x \wedge y> 0$
  - Arrays

    $\forall$ i,j: i>j → select(A, i) > select(A, j)   [array A is sorted]
- The SMT Problem

Quantifier-free Subset

- In **Software Verification**, we will largely restrict ourselves to formulas without quantifiers ($\forall$, $\exists$)—mainly for efficiency reason.
- This is called the quantifier-free fragment of first-order logic.

What you are going to learn this week?

- Solving first order Presburgh formula over integers and rational numbers (Today)
- An efficient procedure for quantifier-free Presburgh formula (Wang)
- Procedure for theory for equality (Tsai)
- Nelson-Oppen SMT procedure (Yu)