0368.4435 Automatic Software Verification April 14, 2015

Lecture 4

Lecturer: Mooly Sagiv *Scribe: Nimrod Busany, Yotam Frank*

Lesson Plan

1. First order logic recap.
2. The SMT decision problem.
3. Basic Verifier Architecture

First-Order Logic

First order logic is a formal notation for mathematics which involves:

- Propositional symbols - $\neg, \vee, \wedge, \rightarrow, \leftrightarrow$
- Predicates - i.e. boolean functions
- Functions - from elements in the underlying domain to an elements in the domain, namely $f(x_1, ..., x_k) = y \mid x_1, ..., x_n, y \in Domain$
- Constant symbols - that are connected to elements in the domain
- Quantifiers - $\forall, \exists$

An example of a first order logic theory:

- Domain - all living beings
- Predicate - isMammal(x), canBreathe(x)
- A formula: $\forall_x isMammal(x) \rightarrow canBirth(x)$

In the example above, we define our domain to be all the living beings. We create a predicate isMammal(), which maps each of the elements in the domain to either true of false (if it is a mammal). Further, we create a predicate canBreathe() which captures the ability of some of the elements in our domain to breathe, denoted by canBreathe(). Then, we claim that all mammals can breathe.

Notice that first order logic is more general than propositional (Boolean) logic. Propositional logic only includes Propositional symbols (logical connectors) and an infinite number of boolean variables. The logic connectors are applied over variables and formulas.

First-Order Logic: Syntax

First order logic is made of a sequence of symbols. Some of the symbols are logical symbols and some are non-logical symbols, also referred as parameters. The logical symbols are the propositional connectives (see previous slide), an infinite number of variables and the quantifiers. The non-logical symbols are functions: such as $f(), g(), +, -, \%, bit-wise\&, ...$; Predicates: $\leq, isString(), isEven(), ...$; and Constants: $0, 1, null, ...$;

A more typical example of a formula in first order logic:
$\forall X : D.p(f(X), X) \Rightarrow \exists Y : D.p(f(g(X, Y)), g(X, Y))$

We may ask if the formula above is satisfiable, this means, "does there exists a domain over which the formula above evaluates to true". We can also ask whether the the following formula is valid, which means "does the fomula hold over all domains and any interpretation of the predicate p and functions f,g.

The answer to the first question is yes, the formula is satisfiable, let us show a signature (i.e. domain and interperation over which it holds):

$D : \{0,1\}$; $f(0) = f(1) = 0$; $g(0,0) = g(0,1) = g(1,0) = g(1,1) = 0; p = (0,0)$

As can be seen, for any variable in the domian, g, will return 0, hence for any x, $|g(X,Y)| = 0$. Therefore, $|f(g(X,Y))| = |f(0)| = 0$; This means that for any x, $p(f(0),0))$ will evaluate to true. Since the right left side of the inference holds for any element in our domain, then the formula will holds for all elements.

The answer to the second question is no, the formula is not valid. To show this, let us find a domain and interperation over which it is violated:

$D : \{0,1\}$; $f(0) = f(1) = 0$; $g(0,0) = g(0,1) = g(1,0) = g(1,1) = 1; p = (0,0)$

Let us look at x=0;the left side of the inference holds, i.e.:
$[|p(f(X),X)|]_{X=0} = |p(f(0),0)| = |p(|f(0)|,0)| = |p(0,0)| = true$
Now let us notice that g, returns 1 for all the elements combinations in our domain. Therefore, the right hand side of the inference:

$[|p(f(g(X,Y)),g(X,Y))|]_{X=0} = |p(f(g(0,Y)),g(0,Y))| = |p(f(|g(0,Y)|),|g(0,Y)|)| = |p(f(1),1|)| = false$ for any Y. Since the left side of the inference hold while the latter does not hold the formula is violated

We have shown a domain and interperation over which the formula is violated hence, it is not valid.

Quantifier-free Subset

I most cases, we would restrict ourselves to a subset of formulas without quantifiers. This set is named the quintifier-free subset or fregemnt of first order logic.

Logical Theory

A logical theory defines a set of parameters (non-logical symbols) with their meanings. We call the definition of the domain and the interpretation of the predicates, functions, and constant as a signature. We have already seen two signatures in the example.

Theory of linear arithmetic
Another example of a signature is the theory of linear arithmetic over the Integers. This theory, uses 0,1 as constants, and gives '+','-' and '$\cdot$' their natural interpretation, the domain of this theory consist of all Integers. This can be represented compactly by: Signature: (0,1,+,-,$\cdot$) over $\mathbb{Z}$

Theory of Presbruger Arithmetic
Another well known theory is the Presbruger Arithmetic. The signature of Presbruger Arithmetic is : Signature: (0,1,+,=) over **Z**
Presbruger Arithmetic consist of a set of axioms:

- $\forall_{X:\mathbb{Z}} \neg((X+1) = 0)$
- $\forall_{X,Y:\mathbb{Z}}((X+1) = (Y+1)) \Rightarrow X = Y$
- $\forall_{X:\mathbb{Z}}(X+0) = X$
- $\forall_{X,Y:\mathbb{Z}}(Y + (X+1)) = (Y + (X+1)))$
- Let $P(X)$ be a first order formula over X:
  $P(0) \vee \forall_{X:\mathbf{Z}} P(X) \Rightarrow P(X+1)) \Rightarrow \forall_{Y:\mathbf{Z}} P(Y)$

Notice that the axioms above capture desired behaviors of the theory. For example, if we add the constant 0, then we require that the result will not change (i.e. $(X + 0) = X$). The third axoim capture the Associative property. While, the last axioms captures the notion of induction.

Many-Sorted Logic - First Order Vocabulary

Many-sorted logic can reflect formally our intention not to handle the universe as a homogeneous collection of objects, but to partition it in a way that is similar to types in typeful programming.

Many sorted logic is equivalent to first order logic, meaning that both systems have the same expressive power. Nevertheless, many sorted logic, is very popular as it lets one easily define types, which ease the use and comprehension of its formulas.

Many sorted logic extends first order logic by defining:

- A finite set of sorts S
- A finite set of function symbols F each with a fixed signature $S^* \to S$
- Constants are defined by zero arity functions
- A finite set of relations symbols R each with a fixed arity S^*

The difference from the first order logic is that here each of the arguments of the constants, functions and predicates is assigned with a sort (type).

In many sorted logic, we split the domain into a set of Sorts S. Formally, $D = \cup_{s \in S} D_s$ (alse, each element in the domain belongs to exacly one of the types). Futher fuctions and relations are defined according to the sorts, formally:

- For every function symbol $f \in F$, an interpretation $\nabla[f]$ assigns the sorts:
  $D_{s_1} \times D_{s_2} \times ... \times D_{s_n} \to D_{s_j}$
- For every relation symbol $r \in R$, an interpretation $\nabla[r]$ assigns the sorts:
  $\nabla[r] \subseteq D_{s_1} \times D_{s_2} \times ... \times D_{s_n}$

Syntax

- Logical variables begin with a capital letter
- Typed Terms <term>::=<variable>| f [(<term>, âĂę <term>)]
- Formulas:
  <formula> ::= <term> = <term> | r(<term>, ... <term>) // atomic
  <formula> ∨ <formula> | <formula> ∧ <formula> | ¬ <formula> // Boolean
  $\exists X : s.$ <formula> | $\forall X : s.$ <formula> //Quantifications

Free variable: The set of free variables in a formula or a term is defined as follows: FV: <term>,<formula> $\to 2^{Var}$

Terms:

- $FV(X) = \{X\}$ // where X is an atomic variable
- $FV(f(< t_1, ..., t_n >) = \cup_{i=1,...,n} FV(t_i)\}$ // where each t_i is a term (possibly complex)

Formulas:

- $FV(t_1 = t_2) = FV(t_1) \cup FV(t_2)$
- $FV(r(< t_1, ..., t_n >) = \cup_{i=1,...,n} FV(t_i)$ // where each t_i is a term
- $FV(f_1 \wedge f_2) = FV(t_1) \cup FV(t_2)$
- $FV(f_1 \vee f_2) = FV(t_1) \cup FV(t_2)$
- $FV(\neg f) = FV(f)$
- $FV(\exists_{x:s} f) = FV(f) - X$ //X is no longer a free variable so we reduce it
- $FV(\forall_{x:s} f) = FV(f) - X$ //X is no longer a free variable so we reduce it

Assignments and Models:

An assignment of the variables is defined by $A : Var \to D$. To apply an assignment over terms we simply perform the assignment by structural induction, formally:
$A(f(t_1, t_2, ..., t_n)) = \nabla[f](A(t_1), A(t_2), ..., A(t_n))$.
Finally, we say that an assignment A models a formula f under interpretation ∇ (denoted by $A, \nabla \models f$) if f is true in A (Notice that this is Tarsky's semantics). Assignment are applied by structural induction as follows:

- $A, \nabla \models t_1 = t_2$ if $A(t_1) = A(t_2)$
- $A, \nabla \models r(t_1, t_2, ..., t_n)$ if $<r(A(t_1), A(t_2), ..., A(t_n)) > \in \nabla[r]$
- $A, \nabla \models f_1 \wedge f_2$ if $A, \nabla \models f_1$ and $A, \nabla \models f_2$
- $A, \nabla \models f_1 \vee f_2$ if $A, \nabla \models f_1$ or $A, \nabla \models f_2$
- $A, \nabla \models \neg f1$ if not $A, \nabla \models f1$
- $A, \nabla \models \exists X : t.f$ if there exists $d \in D_t$ such that $A[X \to d]$ if $A, \nabla \models f1$
- $A, \nabla \models \forall X : t.f$ if for all $d \in D_t$ such that $A[X \to d]$ if $A, \nabla \models f1$

SEE: notice that I added the last bullet, double check

A T-interpretation:

T interpretation is defined similar to interpretation, but also requires that the domain and the interpretations satisfy the theory requirements(axioms)

An example of a T-interpretation is the theory of Linear Arithmetic, let us return to its interpretation and define it further:

- S=int, F=$\{0^0, 1^0, +^2\}$ R=$\{\leq\}$// the exponent represent the number of arguments the function takes

- $D_{int} = \mathbb{Z}$
- Functions:
  $[|0|] = 0$, $[|0|] = 1$ //this are in fact constant, represented by unary function
  $[| + |] = \lambda x, y : int.x + y$
- Relations:
  $[| \leq |] = \lambda x, y : int.x \leq y$

SEE: I fixed the exponent argument over 1

Alternatively, any interpretation which violates the axiom is invalid. E.g. if we use Presburger Arithmetic and interpret the '+' sign incorrectly, e.g. 2+0=7, then this would not be a valid T-interpretation. As in this example the third Preburger axiom is violated

Assignments and T-Models:

An assignment of the variables is defined by $A : Var \to D$. To apply an assignment over terms we simply perform the assignment by structural induction, formally:
$A(f(t_1, t_2, ..., t_n)) = \nabla[f](A(t_1), A(t_2), ..., A(t_n))$.
Finally, we say that an assignment A which models a theory T, T-models a formula under f under interpretation ∇ (denoted by $A, \nabla \models f$) if f is true in A (Notice that this is Tarsky's semantics). Assignment are applied by structural induction as follows:

- $A, \nabla \models_T t_1 = t_2$ if $A(t_1) = A(t_2)$
- $A, \nabla \models_T r(t_1, t_2, ..., t_n)$ if $<r(A(t_1), A(t_2), ..., A(t_n)) >\in \nabla[r]$
- $A, \nabla \models_T f_1 \wedge f_2$ if $A, \nabla \models_T f_1$ and $A, \nabla \models_T f_2$
- $A, \nabla \models_T f_1 \vee f_2$ if $A, \nabla \models_T f_1$ or $A, \nabla \models_T f_2$
- $A, \nabla \models_T \neg f1$ if not $A, \nabla \models_T f1$
- $A, \nabla \models_T \exists X : t.f$ if there exists $d \in D_t$ such that $A[X \to d]$ if $A, \nabla \models_T f1$
- $A, \nabla \models_T \forall X : t.f$ if for all $d \in D_t$ such that $A[X \to d]$ if $A, \nabla \models_T f1$

The Satisfiability modulo theories (SMT) decision problem

The satisfiability modulo theories (SMT) problem is a decision problem for logical formulas with respect to combinations of background theories expressed in classical first-order logic with equality. The input of for an SMT problem is A quantifier-free formula f over a theory T. The output is a boolean answer, which is true iff there exist an T-interpretation ∇ and an assignment $A : FV(f) \to D$ such that $A, \nabla \models_T f$. The complexity of solving the SMT problems depend greatly on the theory and varies from NPC to undecidable.

SEE: - did I get it right?

The following table summaries the decidability complexity for some of the most famous theories:

sym.	Theory	Quantifiers Decidable	QFF Decidable
T_E	Equality	NO	YES
T_{PA}	Peano Arithmetic	NO	NO
$T_{\mathbb{N}}$	Presburger Arithmetic	Yes	YES
$T_{\mathbb{Z}}$	Linear Integer Arithmetic	Yes	YES
$T_{\mathbb{R}}$	Real Arithmetic	Yes	YES
$T_{\mathbb{Q}}$	Linear Rationals	Yes	YES
T_A	Arrays	NO	YES

sym.	Theory	Quantifiers	QF Conjunctive
PL	Propositional Logic	NP-Complete	$O(N)$
T_E	Equality Arithmetic	–	$O(n\log(n))$
$T_{\mathbb{N}}$	Presburger Arithmetic	$O(2\hat{}2\hat{}2\hat{}kn)$	NP-Complete
$T_{\mathbb{Z}}$	Linear Integer Arithmetic	$O(2\hat{}2\hat{}2\hat{}kn)$	NP-Complete
$T_{\mathbb{R}}$	Real Arithmetic	$O(2\hat{}2\hat{}kn)$	$O(2\hat{}2\hat{}kn)$
$T_{\mathbb{Q}}$	Linear Rationals	$O(2\hat{}2\hat{}kn)$	PTIME
T_A	Arrays	-	NP-Complete

*Note - refers input formula size; k - some positive integer;

Basic Verifier Architecture

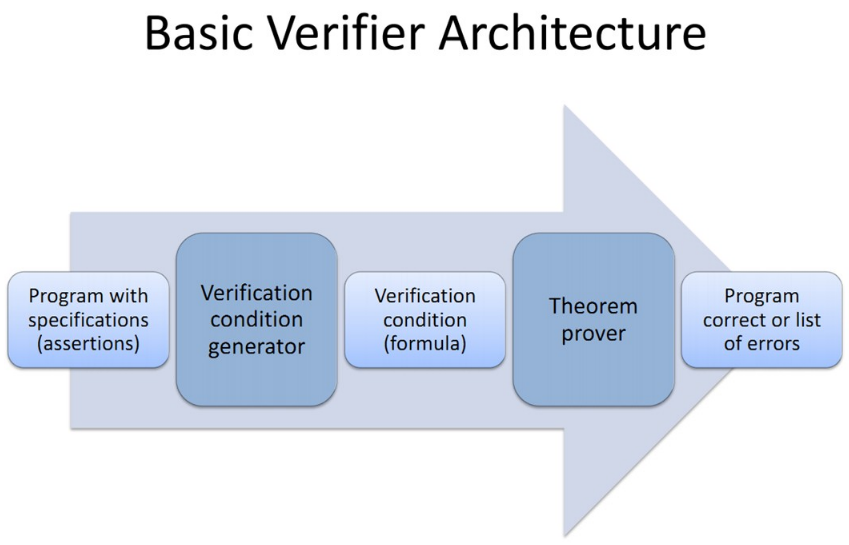
The Basic Verifier has two main components:

1. Verification Condition Generator - Receives Program with specification (assertions etc.) and outputs a set of formulas called Verification Conditions.
2. Theorem Prover - Receives the verification conditions and outputs a 'Program correct' in case it's valid or else list of error ('bugs').

We give a more detailed description of the two components.

Verification Condition Generator

- Creates verification conditions (a set of mathematical logic formulas) from program's source code (VC for short). If VC is valid program is correct. If VC is invalid there might be an error in the program.
- Verification conditions creation is based on the theory of Hoare triplets and they are computed automatically using *'weakest precondition' (wp)*.


Basic Verifier Architecture
Program with specifications (assertions)
Verification condition generator
Verification condition (formula)
Theorem prover
Program correct or list of errors

Simple command Language

`http://meme.nbcr.net/meme/`[1]

References

[1] Bailey, T. L. and C. Elkan, “Fitting a mixture model by expectation maximization to discover motifs in biopolymers.,” *International Conference on Intelligent Systems for Molecular Biology; ISMB.*, pp. 28–36, 1994.