Automated Reasoning for Software Engineering

L. Georgieva and A. Ireland

School of Mathematical and Computer Sciences

Heriot-Watt University

Lilia: Office G.50

Email: lilia@macs.hw.ac.uk

First-order logic (FOL)

- Propositional logic deals only with facts/statements about the world which may or may not be true.
- In FOL variables refer to objects in the world and can be quantified over.
- Examples of statements that can be made in FOL but not in propositional logic: general statements or rules.

FOL syntax

- Term:
  - constant symbols: names.
  - variables: x, y, z.
  - Function symbols applied to one or more terms.

$$F(x), F(F(x)), FatherOf(John)$$

- Sentence: a predicate symbol applied to zero or more terms.

$$on(a, b), sister(Jane, Joan), Jane, itIsRaining(), t_1 = t_2$$

- For a variable v and a sentence Φ:

$$\forall x \Phi, \exists x \Phi$$

  are sentences.

- Closure under $\wedge$, $\vee$, $\leftrightarrow$, $\longrightarrow$, $\neg$.

FOL interpretation

- Interpretation I.
  - U a set of objects, domain of discourse, universe.
  - Maps constant symbols to elements of U.
  - Maps predicate symbols to relations in U (binary relation is a set of pairs).
  - Maps function symbols to functions on U.

Basic FOL semantics

Denotation of terms (naming).

- I(Fred) if Fred is a constant, then given.
- I(x) undefined.
- I(F(term)) I(F)I(term)

$\models_i P(t_1, \ldots, t_n)$ iff $< I(t_1), \ldots, I(t_n) >\in I(P)$ Example: $brother(John, Joe)$?

- $I(John)$ an element of U
- $I(Joe)$ an element of U
- $I(brother) = \{< \ldots, \ldots >, \ldots\}$
- $\models_i brother(John, Joe)$

Semantics of quantifiers

Extend an interpretation I to bind variable x to an element $a \in U : I_{x/a}$.

- $\models_f \forall X.\Phi$ iff $\models_{x/a} \Phi$ for all $a \in U$.
- $\models_f \exists X.\Phi$ iff $\models_{x/a} \Phi$ for some $a \in U$.

Convention: Quantifier applies to the formula to the right until right after enclosing parenthesis.

$$(\forall x.p(x) \vee q(x)) \wedge \exists x.r(x) \longrightarrow q(x)$$

Semantics of PVS

In this lecture we present the formal semantics of the specification language of PVS.

Specification language

- is a media for expressing *what* is computed rather than *how* it is computed.
- shares features with programming languages.
- is a logic in which the behaviour of a computational system can be formalised.

PVS: Applications

- Used for specifying and verifying properties of digital hardware and software systems.
- The PVS language contains constructs that can be statically checked using a theorem prover.
- The logic of PVS is based on simply typed higher-order logic:
  - sybtypes are analogous to subsets.

Orders of a logical system

- Predicates that speak about objects of the domain are first-order.
- Predicates that speak about objects of at most i order are $i + 1$ order.
- Functions that take and return objects of the domain are first-order.
- Functions that take and return objects of at most i order are $i + 1$ order.

Examples of constructs involving higher order

- Induction:

$$\forall P P(0) \wedge \forall n : Nat P(n) \longrightarrow P(n+1) \longrightarrow \forall n : Nat P(n)$$

- Differentiation:

$$(xy)' = x'y + y'x$$

- Statements involving functions: every function that is first-order differentiable in the complex plane is infinitely often differentiable.

- Abstract mathematical structures: lattices, groups.

λ notation

- In usual mathematical notation consequences of functions and formulae are confused.
  - the expression xy^2 can represent infinitely many functions of type $Real \longrightarrow Real$.
  - differentiation is a function of type $(Real \implies Real) \longrightarrow (Real \longrightarrow Real)$: mathematicians speak of differentiation over a variable:
    * differentiation of xy^2 after x results in y^2.
    * differentiation of xy^2 after y results in $2xy$.
    * differentiation of xy^2 after z results in 0.
  - We need notation for functions: $\boldsymbol{\lambda} x : Xt(x)$ is the function that for every $n \in X$ has the value $[t(n)]$.
  - If $[t(x)]$ has the type Y on the assumption that x has the type X then $\boldsymbol{\lambda} x : Xt(x)$ has the type $X \longrightarrow Y$.

Types

- Impose discipline on the specification.
- Lead to easy and early detection of large class of semantical and syntactical errors.
- Useful in mechanised reasoning.

Type definition of the simply typed fragment of PVS

Set U defined cumulatively starting from the base sets 2 and R and including:

- Cartesian products: used to model products in PVS.
- function spaces: used to model function types.
- subsets of previously included sets: used to model predicate subtypes.

$$
\begin{array}{rcl}
U_0 & = & \{2, R\} \\
U_{i+1} & = & U_i \cup \{X \times Y \mid X, Y \in U_i\} \cup \{X^Y \mid X, Y \in U_i\} \\
 & & \bigcup_{x \in U_i} \rho(X) \\
U_\omega & = & \bigcup_{i \in \omega} U_i \\
U & = & U_\omega
\end{array}
$$

Constructors in PVS

- In order to formally reason about mathematical objects, or programs we need a formal language PVS uses higher order logic.
- Constructions in higher order logic used in PVS:
  - $\neg$ not
  - $\wedge$ and
  - $\vee$ or
  - $\longrightarrow$ if ... then
  - $\leftrightarrow$ if and only if
  - $\forall x : XP(x)$
  - $\exists x : xP(x)$
  - $=$ is equal to
  - $p(t_1, \ldots, t_n)$, $t_1, \ldots, t_n$ are in relationship with each other: $(t_1, \ldots, t_n)$ are called atoms;

Examples

- The atoms $p(t_1, \ldots, t_n)$ can have the form:
  - $a < b$
  - $1 < 1 + 1$
  - even(4), odd(5);
- Examples of formulae are:
  - $\forall x, y : \mathrm{Nat} \leftrightarrow x + 1 < y + 1$
  - $\forall x, y : \mathrm{Nat} \leftrightarrow x < y \longrightarrow x < y + 1$
  - $\forall \mathrm{prime}(p) : \mathrm{Nat} \leftrightarrow \neg \exists x : \mathrm{Nat} 1 < x \text{ and } x < p \wedge \mathrm{divides}(x, p)$
  - $\forall x, y : \mathrm{Real\ square}(x + y) = \mathrm{square}(x) + \mathrm{square}(y) + 2 * x * y$

Types in PVS

PVS is a strongly typed specification language:

The simply typed fragment of PVS includes:

- types constructed from the base types by
  - function and product type constructions.
- expressions constructed with constants and variables by
  - application, abstraction, and tupling.

Context, meta variables

Expressions are checked to be well typed under a *context* which is a partial function which assigns a *kind*, i.e. one of (TYPE, CONSTANT, or VARIABLE) to each symbol and a type to the constant and variable symbols.

Notation: Γ, Δ, Θ used for meta variables to range over contexts; A, B, T: variables range over PVS type expressions; r, s range over symbols, identifiers; meta variables x, y range over PVS variables; a, b, f, g range over PVS terms.

Base types are called pretypes.

- function pretype $A \longrightarrow B$
- product pretype $[A, B]$

Preterm: term that has been checked in a context:

$$TRUE, \neg TRUE, \boldsymbol{\lambda}(x : bool) : \neg x$$

Example of context

Context: sequence of declarations

$$s : \mathit{TYPE}, c : T \text{ where T is a type } x : \mathit{varT}$$

Example:

$$\mathit{bool} : \mathit{TYPE} \;\; \mathit{TRUE} : \mathit{bool} \;\; \mathit{FALSE} : \mathit{bool} \;\; x : \mathit{VAR}[[\mathit{bool}, \mathit{bool}] \longrightarrow \mathit{bool}]]$$

Types rules

Given by recursively defined partial function $\boldsymbol{\tau}$ that assigns

- a type $\boldsymbol{\tau}(\Gamma)(a)$ to a preterm a that is well typed wrt a context Γ
- the keyword TYPE as a result of $\boldsymbol{\tau}(\Gamma)(a)$ when A is a well formed type under the context Γ.
- the keyword CONTEXT as the result of $\boldsymbol{\tau}(\Gamma)(\Delta)$ when Δ is a well formed context under the context Γ. For the simply typed fragment Γ is empty.

Type rules in PVS

- The type assignment is deterministic.
- Soundness proof needs to show that the meaning of a term is a meaning of its canonical type.
- The meaning of a term is given by a recursive definition on the term itself.

Type rules

$$\begin{array}{lcl}
\boldsymbol{\tau}()(\{\}) & = & \mathit{CONTEXT} \\
\boldsymbol{\tau}()(\Gamma, s : \mathit{TYPE}) & = & \mathit{CONTEXT}, \text{ if } \Gamma \text{ is undefined,} \\
 & & \boldsymbol{\tau}(\Gamma), (T) = \mathit{TYPE}, \\
 & & \text{ and } \boldsymbol{\tau}()(\Gamma) = \mathit{CONTEXT} \\
\boldsymbol{\tau}()(\Gamma, x : \mathit{VAR}\ T) & = & \mathit{CONTEXT}, \ \mathit{if}\ \Gamma(x)\ \mathit{is\ undefined}. \\
\boldsymbol{\tau}(\Gamma)(T) & = & \mathit{TYPE} \\
\boldsymbol{\tau}()(\Gamma) & = & \mathit{CONTEXT} \\
\boldsymbol{\tau}(\Gamma)(s) & = & \mathit{TYPE}\ \mathit{iff}\ \mathit{kind}(\Gamma(s)) = \mathit{TYPE} \\
\boldsymbol{\tau}(\Gamma)([A \rightarrow B]) & = & \mathit{TYPE}, \ \mathit{if}\ \boldsymbol{\tau}(\Gamma)(A) = \boldsymbol{\tau}(\Gamma)(B) = \mathit{TYPE}
\end{array}$$

Example

Let $\boldsymbol{\omega}$ label the context:

$$\mathit{bool} : \mathit{TYPE}, \mathit{TRUE} : \mathit{bool}, \mathit{FALSE} : \mathit{bool}$$

$$\begin{array}{lcl}
\boldsymbol{\tau}()(\{\}) & = & \mathit{CONTEXT} \\
\boldsymbol{\tau}()(\boldsymbol{\omega}) & = & \mathit{CONTEXT} \\
\boldsymbol{\tau}(\boldsymbol{\omega})([[\mathit{bool}, \mathit{bool}] \longrightarrow \mathit{bool}]) & = & \mathit{TYPE} \\
\boldsymbol{\tau}(\boldsymbol{\omega})(\mathit{TRUE}, \mathit{FALSE}) & = & [\mathit{bool}, \mathit{bool}]
\end{array}$$

Meaning function; definition

Returns the meaning of a well-formed type A and a well formed expression a in the context Γ.

$$
\begin{aligned}
M(\Gamma \mid \gamma)(s) &= \gamma(s) \\
&\quad \text{if } \mathrm{kind}(\gamma(s)) \in \{TYPE, CONSTANT, VARIABLE\} \\
M(\Gamma \mid \gamma)([A \longrightarrow B]) &= M(\gamma \mid \Gamma)B^{M(\Gamma|\gamma)(A)} \\
M(\Gamma \mid \gamma)([T_1, T_2]) &= M(\gamma \mid \Gamma)(T_1) \times M(\Gamma \mid \gamma)(T_2)
\end{aligned}
$$

Meaning function: Example

Example: let ω be an assignment for the context Ω of the form

$$\{bool \leftarrow 2\}\{TRUE \leftarrow 1\}\{FALSE \leftarrow 0\}$$

then

$$M(\Omega \mid \omega)([bool, bool]) = 2 \times 2$$

$$M(\Omega \mid \omega)([TRUE, FALSE]) =< 1, 0 >$$

Satisfaction

A context assignment γ is said to satisfy a context Γ, denoted as $\gamma \models \Gamma$ iff

1. $\gamma(\mathit{bool}) = 2$
2. $\gamma(\mathit{TRUE}) = 1$
3. $\gamma(\mathit{FALSE}) = 0$
4. $\gamma(s) \in U$ whenever $\mathit{kind}(\Gamma(s)) = \mathit{TYPE}$, and
5. $\gamma(s) \in M(\Gamma \mid \gamma)(\mathit{type}(\Gamma(s)))$ whenever $\mathit{kind}(\Gamma(s)) \in \{\mathit{CONSTANT}, \mathit{VARIABLES}\}$

Example: Satisfaction

The assignment $\omega\{one \leftarrow 1\}\{zero \leftarrow 0\}$ satisfies the context:

Ω, $one : TYPE$, $zero : one$.

Typing judgements are not invalidated when the context is extended.

Types in PVS

- A type in PVS is a set of values.
- Questions:
  - Which values contain a given type?
  - How can one construct these values?
  - What purpose do the types serve?

Types in PVS

Basic built in types:

- **bool**: two element set of true values TRUTH and FALSE;
- **nat**: countable set of natural numbers 0, 1, 2 ...;
- **int**: countable set of integers -2, -1, 0, 1, 2 ...;
- **rat**: countable set of rational numbers 1, 0.5 $\frac{1}{3}$,... ;
- **real**: uncountable set of real numbers 1, 0.33, $\frac{1}{\sqrt{3}}$, π;

Constructing types from types

Tuples:

- Type $[T_1, \ldots, T_n]$, $n \geq 1$
- Meaning: Cartesian product $T_1 \times \ldots \times T_n$.
- Constructor: ()
- Destructors $1', \ldots, n'$
- Examples:
  - $[int] = int$
  - (7, TRUE)= [int, bool]
  - $(7, \mathrm{TRUE})^1 = 7$
  - $(7, \mathrm{TRUE})^2 = TRUE$