Automated Reasoning for Software Engineering

L. Georgieva and A. Ireland

School of Mathematical and Computer Sciences

Heriot-Watt University

Lilia: Office G.54

Email: lilia@macs.hw.ac.uk

Higher Order Logic and Sequent Calculus. Introduction

- 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
  - $\rightarrow$ 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:
  - $\forall x, y : \mathrm{Nat} \leftrightarrow x + 1 < y + 1$
  - $\forall x, y : \mathrm{Nat} \leftrightarrow x < y \rightarrow x < y + 1$
  - $\forall \mathrm{prime}(p) : \mathrm{Nat} \leftrightarrow \neg \exists x : \mathrm{Nat} 1 < x \text{ and } x < p \wedge \mathrm{divides}(x, p)$
  - $\forall x, y : \mathrm{Real}\ \mathrm{square}(x + y) = \mathrm{square}(x) + \mathrm{square}(y) + 2 * x * y$

Predicate versus higher order logic: what is an order

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

Example: The induction principle is second order.
$\forall P : Nat \to \mathrm{Bool} P0 \wedge \forall n : Nat(P(n) \to P(n+1)) \to \mathrm{Nat} : P(n)$

Higher-order logic

- When reasoning about physical objects the following principles are considered valid:
  - law of excluded middle: $A \vee \neg A$
  - law of double negation: $\neg\neg A \implies A$

  Example: Either there are errors in the code, or there are no errors in the code.

Intuitionistic or constructive logic

- Mathematical objects.
- The law of excluded middle is not observed.
- To prove $\exists x : Xp(x)$ in Intuitionistic logic means to find a witness t for which $p(t)$ holds.

Sequent calculus for first-order logic

The most important types of deduction systems are:

- natural deduction
  - models the natural style of reasoning;
  - principle of forward reasoning: deriving conclusions, deriving conclusions from the conclusions, etc.
- sequent calculus;
  - conclusions and premises are treated in the same way.
  - the proof consists o judgments rather than conclusions;
- PVS is based on a sequent calculus for higher order classical logic.
- COQ is based on higher order intuitionistic logic with inductive types.

Sequent Calculus for Classical Logic

Definition: A multiset is a set that can distinguish how often en element occurs in it. Alternatively: a list that cannot see the order of its elements.

Examples

1. $A \vee B \;\; A \wedge B \;\; A \wedge B$
2. $A \vee B \;\; A \wedge B \;\; C \implies D$
3. $A \wedge B \;\; A \vee B \;\; A \wedge B$

The first and the last multiset are equal.

Sequents

A sequent is an object of the form:

$$\Gamma \Vdash \Delta$$

where:

- Both Γ and Δ are multisets of formulae.

Meaning: Whenever all of the Γ are true then at least one Δ is true.

Propositional rules

- Axiom:

$$\overline{\Gamma, A \Vdash \Delta, A}$$

- The cut rule:

$$\frac{\Gamma, A \Vdash B \qquad \Gamma \Vdash \Delta, A}{\Gamma \Vdash \Delta}$$

Structural rules

- Weakening (left):

$$\frac{\Gamma \Vdash \Delta}{\Gamma, A \Vdash \Delta}$$

- Weakening (right):

$$\frac{\Gamma \Vdash \Delta}{\Gamma \Vdash \Delta, A}$$

Structural rules (Cntd)

- Contraction (left):

$$\frac{\Gamma, A, A \Vdash \Delta}{\Gamma, A \Vdash \Delta}$$

- Contraction (right):

$$\frac{\Gamma \Vdash \Delta, A, A}{\Gamma \Vdash \Delta, A}$$

Rules for the constants

- ($\top$ left):

$$\frac{\Gamma \Vdash \Delta}{\Gamma, \top \Vdash \Delta}$$

- ($\bot$-left):

$$\frac{}{\Gamma, \bot \Vdash \Delta}$$

- ($\top$ right):

$$\frac{}{\Gamma \Vdash \Delta, \top}$$

- ($\bot$-right):

$$\frac{\Gamma \Vdash \Delta}{\Gamma \Vdash \Delta, \bot}$$

Rules for negation

- Negation (left):

$$\frac{\Gamma \Vdash \Delta, A}{\Gamma, \neg A \Vdash \Delta}$$

- Negation (right):

$$\frac{\Gamma, A \Vdash \Delta}{\Gamma \Vdash \Delta, \neg A}$$

Rules for Conjunction and Disjunction

- ($\wedge$ left):

$$\frac{\Gamma, A, B \Vdash \Delta}{\Gamma, A \wedge B \Vdash \Delta}$$

- ($\vee$-left):

$$\frac{\Gamma, A \Vdash \Delta \quad \Gamma, B \Vdash \Delta}{\Gamma, A \vee B \Vdash \Delta}$$

- ($\wedge$ right):

$$\frac{\Gamma \Vdash \Delta, A \quad \Gamma \Vdash \Delta, B}{\Gamma \Vdash \Delta, A \wedge B}$$

- ($\vee$-right):

$$\frac{\Gamma \Vdash \Delta, A, B}{\Gamma \Vdash \Delta, A \vee B}$$

Premises and conclusions are treated in the same way.

Rules for $\rightarrow$ and $\leftrightarrow$

- ($\rightarrow$-left):

$$\frac{\Gamma \Vdash \Delta, A \quad \Gamma, B \Vdash \Delta}{\Gamma, A \rightarrow B \Vdash \Delta}$$

- ($\rightarrow$-right):

$$\frac{\Gamma, A \Vdash \Delta, B}{\Gamma \Vdash \Delta, A \rightarrow B}$$

- ($\leftrightarrow$-left):

$$\frac{\Gamma, A \Vdash B, A \quad B \rightarrow A, \Vdash \Delta}{\Gamma, A \leftrightarrow B \Vdash \Delta}$$

- ($\leftrightarrow$-right):

$$\frac{\Gamma \Vdash \Delta, A \rightarrow B \quad \Gamma \Vdash \Delta, B \rightarrow A}{\Gamma \Vdash \Delta, A \leftrightarrow B}$$

Rules for the quantifiers

- ($\forall$-left):

$$\frac{\Gamma, P[x := t] \Vdash \Delta}{\Gamma, \forall x : X P(x) \Vdash \Delta}$$

- ($\exists$-left):

$$\frac{\Gamma, P[x := y] \Vdash \Delta}{\Gamma, \exists x : X P(x) \Vdash \Delta}$$

- ($\forall$-right):

$$\frac{\Gamma \Vdash \Delta, P[x := y]}{\Gamma \Vdash \Delta, \forall x : X P(x)}$$

- ($\exists$-right):

$$\frac{\Gamma \Vdash \Delta, P[x := t]}{\Gamma \Vdash \Delta, \exists x : X P(x)}$$

The t is an arbitrary term of type X and X is not free in Γ, Δ

Rules for equality

- Reflection:

$$\frac{}{\Gamma \Vdash \Delta, t = t}$$

- Replication:

$$\frac{t_1 = t_2, \Gamma[t_2] \Vdash \Delta[t_2]}{\Gamma[t_1] \Vdash \Delta[t_1]}$$

Rules for IF

- PVS has an IF operator:
- The operator is defined as $(A \wedge B) \vee \neg A \wedge C$
- IF-left

$$\frac{\Gamma, A, B \Vdash \Delta \quad \Gamma, \neg A, C \Vdash \Delta}{\Gamma, IF(A, B, C) \Vdash \Delta}$$

- IF-right

$$\frac{\Gamma, A \Vdash \Delta, B \quad \Gamma, \neg A \Vdash \Delta, C}{\Gamma \Vdash \Delta IF(A, B, C)}$$

Overview of the type systems in PVS

Types in programming languages:

- Provide structure;
- Provide specification with documents;
- Are naturally allocated with functions;
- Ensure certain correctness properties (e.g. strong normalisation of typed Λ calculus.

Types in logic

- Many logics are untyped (e.g. propositional logic);
- Type specification is convenient (automatically checkable);
- Type specification is expressible in the logic itself;
- In type theory there is a close link between types and logic (PVS is not based on type theory).

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, TRUE)'1=7
  - (7, TRUE)'2=TRUE

We will study:

- Interactive proof tools (proof assistants):
  - offer to prove theorems step by step;
  - user has to select an appropriate command;
  - each step that the prover offers is logically sound;
  - granularity varies;
- Theory behind higher order theorem provers:
  - deductive calculi
  - data types;
  - typed lambda calculus versus higher order logic;
- Applications of PVS to small functional programs.