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

Induction Part 2. PVS by Examples

- Motivation.
- Axioms for natural numbers.
- Mathematical Induction (weak induction).
- Complete induction (strong induction).
- Application: correctness of loops.

Proofs by Induction: Natural numbers

We want to prove the property of monotonicity of addition of natural numbers:

$$\forall k, m, n \in \mathbb{N} : m \leq n \implies m + k \leq n + k$$

- Proof by hand.
- Corresponding PVS theory.
- Proof rules for induction.

Corresponding PVS Theory

```
nat_xmpls: THEORY
  BEGIN

  k, m, n: VAR nat

  linear: THEOREM m <= n IMPLIES m + k <= n + k

END nat_xmpls
```

- Proving theorem *linear*: prove, x-prove.
- Solved by built in decision procedure: *grind*.

Example 2: Natural numbers

- We want to prove that $\forall m, n \in \mathbb{N}\ m \leq n \implies m^2 \leq n^2$.
- In PVS

```
nat_xmpls1: THEORY
  BEGIN


  m, n: VAR nat
 quadratic: THEOREM m <= n IMPLIES m * m <= n * n
END nat_xmpls
```

- Sequent to prove

```
|-------
{1} FORALL(m,n:nat):m<=n IMPLIES m*m<=n*n
```

- Rule *grind* does not prove the sequent.

Example 2: Continued

- Sequent to prove

```
|-------
{1} FORALL(m,n:nat):m<=n IMPLIES m*m<=n*n
```

- Step 1: Eliminate quantifiers with rule (skolem!)
- New sequent:

```
|-------
{1} m!1<=n!1 IMPLIES m!1*m!1<=n!1*n!1
```

- Here m!1 and n!1 are Skolem constants.

Example2: Continued

- Eliminate implication with rule *flatten*.
- New sequent:

  ```
  {-1}m!1<=n!1
  |-------
  {1}m!1*m!1<=n!1*n!1
  ```

- Step 3: Apply the prelude lemma

  `le_times_le_pos`

  with rule

  `(‘‘le_times_le_pos’’)`

  where lemma

  `le_times_le_pos`

  is

$$\forall x, z \in \mathbb{R}_{\geq 0}, \forall y, w \in \mathbb{R} : x \leq y \wedge z \leq w \implies xz \leq yw$$

Alternative Proofs

Other ways to prove $\forall m, n \in \mathbb{N}\ m \leq n \implies m^2 \leq n^2$.

- using a library of rewrite lemmas containing

  `(‘‘le_times_le_pos’’)`

  1. `(auto-rewrite-theory ‘‘real_props’’)`
  2. (grind)

- by induction on n:

  1. `(induct(‘‘n’’))`
  2. (grind)
  3. (grind)

- with more automation

  1. `(‘‘induct-and-simplify ‘‘n’’)`

Recursive functions

```
f(x1:T1,...xn:Tn): RECURSIVE T= expr
MEASURE M
by <<
```

Assuming that

- `<<: [M, M -> bool]`

  is a *well founded* relation;

- `m: [x1: T1,..., xn: Tn ->M]`
- The measure m of each recursive call in expr decreases according to relation $<<$.

Well founded relation

- `<<: [T, T-> bool]`

  is a well founded relation iff every non empty set (in T) has a minimal element.

- PVS inductive sets are based on fixpoint theory. General fixpoint theory considers a complete partial order structure $(E, <)$ where
  - E is partially ordered by $<$, meaning that $<$ is a reflexive, transitive and antisymmetric relation in E;
  - E has a bottom (least) element, often denoted by $\perp$
  - Every chain c in E has a least upper bound (lub) in E, where a chain c in E is a totally ordered subset of E.

PVS inductive sets

- E correspond to a type of the form [T -> bool], whose elements are all sets of elements of type T;
- The partial order $<$ is set inclusion, denoted subset[T] in sets PVS prelude theory.
- The bottom element is $emptyset[T]$;
- The existence of a lub for a chain of sets, or any set of sets, is satisfied, since every set of sets admits its union for lub;
- Fixpoint induction is useful for proving that all elements in $mu(tau)$ satisfy some property p, in other words, that $mu(tau)$ is a subset of p:

```
(FORALL (x: T):  tau(p)(x)  IMPLIES  p(x))
-------------------------------------------
(FORALL (x: T):  mu(tau)(x)  IMPLIES  p(x))
```

- PVS generates as an axiom a specialized version of the above, for each INDUCTIVE definition of a set.

Recursive functions: Example

Recursive definition of $\sum_{i=0}^{n} i$ in PVS:

```
sum(n): RECURSIVE nat =
  (IF n = 0 THEN 0 ELSE n + sum(n - 1) ENDIF)
  MEASURE (LAMBDA n: n)
```

- sum is a function from nat to nat.
- MEASURE is a function from arguments of sum, here (nat) to a well-founded domain (here nat).
- Termination of sum is ensured because MEASURE is strictly decreasing for each recursive call (checked automatically).

Recursive functions. Continued

Prove that $\Sigma_{i=0}^{n} i = \frac{n*(n+1)}{2}$

- Proof by hand.
- Proof in PVS

```
sum_form: THEOREM sum(n) = (n * (n + 1))/2
```

- Proof:

```
induct_and_simplify(``n'')
```

Axioms in PVS

Unlike theorems, axioms do not have to be proved!

PVS generates implicit axioms from

- recursive definitions,
- inductive definitions,
- abstract datatype definitions.

Soundness of such axioms is PVS responsibility.

A user should declare theorems or lemmas instead of axioms.

Induction in PVS

Homework: work through the examples of proving properties of natural numbers in PVS as explained in Chapter 2 of the system guide which is available from the exercise directory of the module website.

Natural Deduction Proofs: Examples

Check lecture 4 for the rules of natural deduction and for more examples that you may want to try out in PVS.

$$P \longrightarrow (Q \longrightarrow R) \vdash (P \land Q) \longrightarrow R$$

1	$P \longrightarrow (Q \longrightarrow R)$	*premise*
2	$P \land Q$	*assumption*(8)
3	P	$\land e(2)$
4	Q	$\land e(2)$
5	$Q \longrightarrow R$	$\longrightarrow e(1)$
6	R	$\longrightarrow e(5)$
7	R	what we need
8	$(P \land Q) \longrightarrow R)$	