ECE 741/841

19 September 2002

Free Variables in the Body of Function

In the last lecture, we saw that a free variable is not permitted in the body of a function declaration.

```
exercise1 : theory
begin

a,b : VAR bool

x,y : VAR real

P(a) : bool = x > 5

end exercise1
```

This is because a function, given its arguments, must return a unique value.

Proving in PVS

In the last lecture we declared a theory and within the theory we declared constants, variables, types, predicates, and functions in the PVS system.

We can also declare Axioms, Lemmas, and Theorems.

Axioms are statements that are taken as true without a proof. Axioms should be avoided because they can introduce inconsistencies in the theory.

Lemmas and Theorems can be declared, but the are hypothesis until a proof has been constructed.

To declare a Theorem (or lemma), a name (identifier) is declared of type theorem and then the theorem is stated.

Theorem (Lemma) Declaration

```
exercise2 : theory

begin

n,m : VAR nat

th1 : theorem
forall n,m : n > m => n+1 > m

end exercise2
```

The Proof Interaction

After we declare a theorem, we can start an interactive session with PVS to construct a proof of the theorem.

This is done by putting the cursor anywhere over the theorem declaration and type the sequence.

```
M-x pr
```

or

```
M-x prove
```

After this command is typed, the system will type check the theory (if it is not already) and start an interaction buffer.

The Sequent

The interaction buffer shows a sequent, and asks for a rule.

The sequent is a list of all the premises, and all the formulas you are attempting to prove.

Example:

$$\mathrm{P1(n),P2(n),P3(n),P(4)} \vdash \mathrm{Q1(n),Q2(n),Q3(n),Q4(n)}$$

Which is the same as,

$$\mathrm{P1(n)} \wedge \mathrm{P2(n)} \wedge \mathrm{P3(n)} \wedge \mathrm{P(4)} \vdash \mathrm{Q1(n)} \vee \mathrm{Q2(n)} \vee \mathrm{Q3(n)} \vee \mathrm{Q4(n)}$$

The Sequent, cont.

In PVS, the sequent will look like this:

```
{-1}  P1(n)
{-2}  P2(n)
{-3}  P3(n)
{-4}  P4(n)
  |-------
{1}   Q1(n)
{2}   Q2(n)
{3}   Q3(n)
{4}   Q4(n)
```

The Sequent, cont.

Lets declare a theorem of that form:

```
exercise2 : theory
begin

n,m : VAR nat
P1,P2,P3,P4 : [nat -> bool]
Q1,Q2,Q3,Q4 : [nat -> bool]

th2 : theorem
P1(n) AND P2(n) AND P3(n) AND P4(n) =>
             Q1(n) OR Q2(n) OR Q3(n) OR Q4(n)

end exercise2
```

Interaction

If we put the cursor on the theorem declaration an type `M-x pr`, a proof session is started:

```
  | -------
{1}   FORALL (n: nat):
        P1(n) AND P2(n) AND P3(n) AND P4(n) =>
         Q1(n) OR Q2(n) OR Q3(n) OR Q4(n)

Rule?
```

Now we can type `(skolem 1 "np")` and that will get rid of the universal quantifier and replace the variable `n` with `np`. We can also type `(skolem!)` and the system will pick arbitrary variables for us.

After the (skolem 1 "np") Command

```
Rule? (skolem 1 "np")
For the top quantifier in 1, we introduce Skolem constants: np,
this simplifies to:
th2 :

  |-------
{1}   P1(np) AND P2(np) AND P3(np) AND P4(np) =>
       Q1(np) OR Q2(np) OR Q3(np) OR Q4(np)

Rule?
```

Now we can type `flatten` and this rule will move the antecedent of the implication to the premises list and break conjunctions in the antecedent and disjunctions in the conclusion.

After (flatten)

```
Rule? (flatten)
Applying disjunctive simplification to flatten sequent,
this simplifies to:
th2 :

{-1}  P1(np)
{-2}  P2(np)
{-3}  P3(np)
{-4}  P4(np)
  |-------
{1}   Q1(np)
{2}   Q2(np)
{3}   Q3(np)
{4}   Q4(np)
```

Instantiation

Instantiation allows the user to supply a witness for the proof.

Examples:

$\exists n : n > 4$

$(\forall n : P(n)) \Rightarrow P(17)$

Important!

You can instantiate universally quantified formulas in the premises and existentially quantified formulas in the conclusion (goal).

You cannot instantiate universally quantified formulas in the goal or existentially quantified formulas in the premises.

My First Mechanized Theorems

Lets now prove the theorems from the homework.

$\forall x : \exists y : y > x$

It is false that $\exists x : \forall y : y > x$

Other Commands

`(undo n)` moves the proof one step backwards.

`(assert)` applies algebraic simplification.

`(expand "f" 1)` expands the definition of function f in formula 1

`(replace 1 2)` replaces formula 1 into formula 2

Recursive Definitions

It is possible to define a function that invoke itself. This is a recursive definition. Such a definition cannot be infinite; it must terminate.

Example:

```
fact(n) : recursive nat =
if n = 0 then 1 else n*fact(n-1) endif
measure n;
```

Executability of Functions

Note that this is not a programming language. Definitions are not executable in the programming sense. However, we can unfold a definition to show that the value of a function applied to a concrete value is the value that we expect.

Example:

$\mathrm{fact}(5) = 120$

Homework

Define a recursive function in PVS that is a summation of numbers up to n.

That is:

$$\sum_{i=1}^{n} i$$