I217: Functional Programming

14. Proof Assistant

Kazuhiro Ogata, Canh Minh Do

Roadmap

- CafeOBJ CITP: Proof Assistant for CafeOBJ
- Associativity of _+_
- Commutativity of _+_
- Associativity of _*_
- Commutativity of _*_
- Correctness of a Tail Recursive Factorial
- Associativity of _@_
- Correctness of a Tail Recursive Reverse

CafeOBJ CITP:Proof Assistant for CafeOBJ

CafeOBJ is equipped with a proof assistant called *CafeOBJ CITP* that supports to conduct theorem proving.

Writing proof scores manually, existing constants may be used as fresh constants – *human errors*.

CafeOBJ CITP can reduce such human errors.

CafeOBJ CITP provides a set of commands for supporting to conduct theorem proving.

Proofs written by using such commands are called proof scripts.

Proof Assistant for CafeOBJ

The commands used in this course:

:goal { *eqs* } where *eqs* is a set of (conditional) equations (senteces) to prove

The label of the equation

E.g. **:goal** { **eq** [assoc+] : (X + Y) + Z = X + (Y + Z) . }

A module *Spec* is supposed to be opened.

An initial current goal $Spec \vdash eqs$ (that *eqs* is derived or proved from *Spec*) is defined.

:ind on *X*:*S* where *X* is a variable of a (constrained) sort *S*

A variable *X* is specified to which (simultaneous) structural induction is applied.

Proof Assistant for CafeOBJ

:apply (si)

(Simultaneous) structural induction is applied to the current goal on the variable X specified with **:ind on**, replacing the current goal with n sub-goals, where n is the number of the constructors of the sort S of X, and introducing fresh constants for the non-constant constructors.

:apply (tc)

Each variable in the sentences of the current goal is replaced with a fresh constant. If the current goal has two or more sentences, say n, then the goal is replaced with n sub-goals.

:apply (rd)

The sentence to be proved in the current goal is reduced. If it reduces to true, then the current goal is discharged.

Proof Assistant for CafeOBJ

:show proof

An outline of the proof conducted so far is shown. The current goal is marked with >.

:desc proof

The sub-goals generated so far are shown.

:desc .

The current sub-goal is shown.

Associativity of _+_

```
mod! PNAT1 {
  [PNat]
  op 0 : -> PNat {constr} .
  op s : PNat -> PNat {constr} .
  vars X Y Z : PNat .
  eq (0 = s(Y)) = false .
  eq (s(X) = s(Y)) = (X = Y) .
  op _+_ : PNat PNat -> PNat .
  eq 0 + Y = Y .                  -- (+1)
  eq s(X) + Y = s(X + Y) .        -- (+2)
  op _*_ : PNat PNat -> PNat .
  eq 0 * Y = 0 .                  -- (*1)
  eq s(X) * Y = (X * Y) + Y .     -- (*2)
```


Associativity of _+_

```
  op fact1 : PNat -> PNat .
  eq fact1(0) = s(0) .                       -- (f1-1)
  eq fact1(s(X)) = s(X) * fact1(X) .         -- (f1-2)
  op fact2 : PNat -> PNat .
  op sfact2 : PNat PNat -> PNat .
  eq fact2(X) = sfact2(X,s(0)) .             -- (f2)
  eq sfact2(0,Y) = Y .                       -- (sf2-1)
  eq sfact2(s(X),Y) = sfact2(X,s(X) * Y) .   -- (sf2-2)
}
```

Associativity of _+_

```
"Theorem 1. [_+_ is associative (assoc+)]
(X + Y) + Z = X + (Y + Z)
Proof. By induction on X."
open PNAT1 .
:goal { eq [assoc+] : (X + Y) + Z = X + (Y + Z) . }
:ind on (X:PNat)
:apply (si)
-- I. Base case
:apply (tc)
:apply (rd)
-- II. Induction case
:apply (tc)
:apply (rd)
close
"End of Proof of Theorem 1"
```


Associativity of _+_

```
open PNAT1 .
:goal { eq [assoc+] : (X + Y) + Z = X + (Y + Z) . }
```

The equation to be proved is set.

```
:desc .
```

This is the goal to be proved.

```
:goal { ** root ------------------------------------------
  -- context module: %
  -- sentence to be proved
    eq [assoc+]: (X:PNat + Y:PNat) + Z:PNat
      = X + (Y + Z) .
}
```

Associativity of _+_

:ind on (X:PNat)
:apply (si)

X is set as the induction variable.

Structural induction is applied to the induction variable, generating two goals: one for the base case and the other for the induction case.

:desc .

This is the goal for the base case.

```
[si]=>
:goal { ** 1 -----------------------------------------
  -- context module: %
  -- induction variable
    X:PNat
  -- sentence to be proved
    eq [assoc+]: (0 + Y:PNat) + Z:PNat = 0 + (Y + Z) .
}
```


Associativity of _+_

:apply (tc)

Variables are replaced with fresh constants.

:desc .

```
[tc]=>
:goal { ** 1-1 -----------------------------------------
  -- context module: %
  -- induction variable
    X:PNat
  -- introduced constants
    op Y@PNat : -> PNat { prec: 0 }
    op Z@PNat : -> PNat { prec: 0 }
  -- sentence to be proved
    eq [TC assoc+]: (0 + Y@PNat) + Z@PNat
       = 0 + (Y@PNat + Z@PNat) .
}
```

Associativity of _+_

:apply (rd) The first goal is discharged.

:desc . This is the goal for the induction case.

```
[si]=>
:goal { ** 2 ------------------------------------------
  -- context module: %
  -- induction variable
    X:PNat
  -- constant for induction
    op X#PNat : -> PNat { prec: 0 }
  -- introduced axiom
    eq [SI assoc+]: (X#PNat + Y:PNat) + Z:PNat
      = X#PNat + (Y + Z) .
  -- sentence to be proved
    eq [assoc+]: (s(X#PNat) + Y:PNat) + Z:PNat
      = s(X#PNat) + (Y + Z) .
}
```

The induction hypothesis

Associativity of _+_

:apply (tc) Variables are replaced with fresh constants.

:desc .

```
[tc]=>
:goal { ** 2-1 ------------------------------------------
  -- context module: %
  -- induction variable
    X:PNat
  -- introduced constants
    op Y@PNat : -> PNat { prec: 0 }
    op Z@PNat : -> PNat { prec: 0 }
  -- constant for induction
    op X#PNat : -> PNat { prec: 0 }
  -- introduced axiom
    eq [SI assoc+]: (X#PNat + Y:PNat) + Z:PNat
      = X#PNat + (Y + Z) .
  -- sentence to be proved
    eq [TC assoc+]: (s(X#PNat) + Y@PNat) + Z@PNat
      = s(X#PNat) + (Y@PNat + Z@PNat) .
}
```

Associativity of _+_

:apply (**rd**) The second goal is discharged.

Then, the main goal is discharged.

We have proved that _+_ is associative.

Commutativity of _+_

```
"Lemma 1. [Right zero of _+_ (rz+)]
X + 0 = X
Proof. By induction on X."
open PNAT1 .
:goal { eq [rz+] : X + 0 = X . }
:ind on (X:PNat)
:apply (si)
-- I. Base case
:apply (rd)
-- II. Induction case
:apply (rd)
close
"End of Proof of Lemm 1"
```

Commutativity of _+_

```
"Lemma 2. [Right successor of _+_ (rs+)]
X + s(Y) = s(X + Y)
Proof. By induction on X."
open PNAT1 .
:goal { eq [rs+] : X + s(Y) = s(X + Y) . }
:ind on (X:PNat)
:apply (si)
-- I. Base case
:apply (tc)
:apply (rd)
-- II. Induction case
:apply (tc)
:apply (rd)
close
"End of Proof of Lemma 2"
```


Commutativity of _+_

To use the two lemmas that have been proved in the proof that _+_ is commutative, the following module is prepared:

```
mod! PNAT1-RZRS+ {
  pr(PNAT1)
  vars X Y : PNat .
  eq X + 0 = X .
  eq X + s(Y) = s(X + Y) .
}
```

Commutativity of _+_

```
"Theorem 2. [Commutativity of _+_ (comm+)]
X + Y = Y + X
Proof. By induction on X."
open PNAT1-RZRS+ .
:goal { eq [comm+] : X + Y = Y + X . }
:ind on (X:PNat)
:apply (si)
-- I. Base case
:apply (tc)
:apply (rd)
-- II. Induction case
:apply (tc)
:apply (rd)
close
"End of Proof of Theorem 2"
```


Associativity of _*_

Since we have proved that _+_ is associative and commutative, we give _+_ the operator attributes **assoc** and **comm**.

```
mod! PNAT2 {
  ...
  op _+_ : PNat PNat -> PNat {assoc comm} .
  ...
}
```

Associativity of _*_

```
"Lemma 4 [distributive law of _*_ over _+_ (d*o+)]
(X + Y) * Z = (X * Z) + (Y * Z)
Proof. By induction on X."
open PNAT2 .
:goal { eq [d*o+] : (X + Y) * Z = (X * Z) + (Y * Z) . }
:ind on (X:PNat)
:apply (si)
-- I. Base case
:apply (tc)
:apply (rd)
-- II. Induction case
:apply (tc)
:apply (rd)
close
"End of Proof of Lemma 4"
```


Associativity of _*_

To use the lemma that has been proved in the proof that _*_ is associative, the following module is prepared:

```
mod! PNAT2-D*O+ {
  pr(PNAT2)
  vars X Y Z : PNat .
  eq (X + Y) * Z = (X * Z) + (Y * Z) .
}
```

Associativity of _*_

```
"Theorem 3 [Associativity of _*_ (assoc*)]
(X * Y) * Z = X * (Y * Z)
Proof. By induction on X."
open PNAT2-D*O+ .
:goal { eq [assoc*] : (X * Y) * Z = X * (Y * Z) . }
:ind on (X:PNat)
:apply (si)
-- I. Base case
:apply (tc)
:apply (rd)
-- II. Induction case
:apply (tc)
:apply (rd)
close
"End of Proof of Theorem 3"
```


Commutativity of _*_

```
"Lemma 4 [Right zero of _*_ (rz*)]
X * 0 = 0
Proof. By induction on X."
open PNAT2 .
:goal { eq [rz*] : X * 0 = 0 . }
:ind on (X:PNat)
:apply (si)
-- I. Base case
:apply (rd)
-- II. Induction case
:apply (rd)
close
"End of Proof of Lemma 4"
```

Commutativity of _*_

```
"Lemma 5 [Right successor of _*_ (rs*)]
X * s(Y) = (X * Y) + X
Proof. By induction on X."
open PNAT2 .
:goal { eq [rs*] : X * s(Y) = (X * Y) + X . }
:ind on (X:PNat)
:apply (si)
-- I. Base case
:apply (tc)
:apply (rd)
-- II. Induction case
:apply (tc)
:apply (rd)
close
"End of Proof of Lemma 5"
```


Commutativity of _*_

To use the two lemmas that have been proved in the proof that _*_ is commutative, the following module is prepared:

```
mod! PNAT2-RZRS* {
  pr(PNAT2)
  vars X Y : PNat .
  eq X * 0 = 0 .
  eq X * s(Y) = (X * Y) + X .
}
```

Commutativity of _*_

```
"Theorem 4. [Commutativity of _*_ (comm*)]
X * Y = Y * X
Proof. By induction on X."
open PNAT2-RZRS* .
:goal { eq [comm*] : X * Y = Y * X . }
:ind on (X:PNat)
:apply (si)
-- I. Base case
:apply (tc)
:apply (rd)
-- II. Induction case
:apply (tc)
:apply (rd)
close
"End of Proof of Theorem 4"
```


Correctness of a Tail Recursive Factorial

Since we have proved that _*_ is associative and commutative, we give _*_ the operator attributes **assoc** and **comm**.

```
mod! PNAT3 {
  …
  op _*_ : PNat PNat -> PNat {assoc comm} .
  …
}
```

The proof of the next lemma needs Lemma 4, and then the following module is prepared:

```
mod! PNAT3-D*O+ {
  pr(PNAT3)
  vars X Y Z : PNat .
  eq (X + Y) * Z = (X * Z) + (Y * Z) .
}
```

Correctness of a Tail Recursive Factorial

```
"Lemma 6 [Property of sfact2 (sf2-p)]
Y * sfact2(X,Z) = sfact2(X,Y * Z)
Proof. By induction on X."
open PNAT3-D*O+ .
:goal { eq [sf2-p] : Y * sfact2(X,Z) = sfact2(X,Y * Z) . }
:ind on (X:PNat)
:apply (si)
-- I. Base case
:apply (tc)
:apply (rd)
-- II. Induction case
:apply (tc)
:apply (rd)
close
"End of Proof of Lemma 6"
```


Correctness of a Tail Recursive Factorial

To use the lemma that has been proved in the proof that the tail recursive factorial is correct, the following module is prepared:

```
mod! PNAT3-SF2-P {
  pr(PNAT3)
  vars X Y Z : PNat .
  eq [sf2-p] : Y * sfact2(X,Z) = sfact2(X,Y * Z) .
}
```

Correctness of a Tail Recursive Factorial

```
"Theorem 5 [Correctness of tail recursive fact (ctrf)]
fact1(X) = fact2(X)
Proof. By induction on X."
open PNAT3-SF2-P .
:goal { eq [ctrf] : fact1(X) = fact2(X) . }
:ind on (X:PNat)
:apply (si)
-- I. Base case
:apply (rd)
-- II. Induction case
:apply (rd)
close
"End of Proof of Theorem 5"
```


Associativity of _@_

```
mod! LIST1 (E :: TRIV) {
 [List]
 op nil : -> List {constr}
 op _|_ : Elt.E List -> List {constr} .
 op _@_ : List List -> List .
 op rev1 : List -> List .
 op rev2 : List -> List .
 op sr2 : List List -> List .
 vars E E2 : Elt.E .
 vars L1 L2 L3 : List .
 eq (nil = E | L1) = false .
 eq (E | L1 = E2 | L2) = (E = E2) and (L1 = L2) .
```

Associativity of _@_

```
eq nil @ L2 = L2 .                          -- (@1)
eq (E | L1) @ L2 = E | (L1 @ L2) .      -- (@2)
eq rev1(nil) = nil .                        -- (r1-1)
eq rev1(E | L1) = rev1(L1) @ (E | nil) .  -- (r1-2)
eq rev2(L1) = sr2(L1,nil) .               -- (r2)
eq sr2(nil,L2) = L2 .                     -- (sr2-1)
eq sr2(E | L1,L2) = sr2(L1,E | L2) .     -- (sr2-2)
}
```


Associativity of _@_

```
"Theorem 7. [Associativity of _@_ (assoc@)]
(L1 @ L2) @ L3 = L1 @ (L2 @ L3)
Proof. By induction on L1."
open LIST1 .
:goal { eq [assoc@] : (L1 @ L2) @ L3 = L1 @ (L2 @ L3) .}
:ind on (L1:List)
:apply (si)
-- I. Base case
:apply (tc)
:apply (rd)
-- II. Induction case
:apply (tc)
:apply (rd)
close
"End of Proof of Theorem 7"
```

Correctness of a Tail Recursive Reverse

Since we have proved that _@_ is associative, we give _@_ the operator attribute **assoc**.

```
mod! LIST2 {
  ...
  op _@_ : List List -> List {assoc} .
  ...
}
```


Correctness of a Tail Recursive Reverse

```
"Lemma 8 [Property of sr2 (sr2-p)]
sr2(L1,E2 | L2) = sr2(L1,nil) @ (E2 | L2)
Proof. By induction on L1."
open LIST2 .
:goal { eq [sr2-p] : sr2(L1,E2 | L2) = sr2(L1,nil) @ (E2 | L2) . }
:ind on (L1:List)
:apply (si)
-- I. Base case
:apply (tc)
:apply (rd)
-- II. Induction case
:apply (tc)
:apply (rd)
close
"End of Proof of Lemma 8"
```

Correctness of a Tail Recursive Reverse

To use the lemma that has been proved in the proof that the tail recursive reverse is correct, the following module is prepared:

```
mod! LIST2-SR2-P {
  pr(LIST2)
  vars E E2 : Elt.E .
  vars L1 L2 : List .
  eq [sr2-p] : sr2(L1,E2 | L2) = sr2(L1,nil) @ (E2 | L2) .
}
```


Correctness of a Tail Recursive Reverse

```
"Theorem 8. [Correctness of a tail recursive rev (ctrr)]
rev1(L1) = rev2(L1)
Proof. By induction on L1."
open LIST2-SR2-P .
:goal { eq [ctrr] : rev1(L1) = rev2(L1) . }
:ind on (L1:List)
:apply (si)
-- I. Base case
:apply (rd)
-- II. Induction case
:apply (rd)
close
"End of Proof of Theorem 8"
```

Exercises

1. Type all proofs as well as all specifications in the lecture note and feed them into CafeOBJ.

2. Make comparison of manual proofs, proof scores in CafeOBJ and proof scripts for CafeOBJ CITP, writing your opinions on the three approaches to program verification.

Exercises

3. Investigate CafeInMaude, CafeInMaude Proof Assistant (CiMPA), CafeInMaude Proof Generator (CiMPG, CIMPG+F).

https://doi.org/10.1145/3208951

https://doi.org/10.1016/j.jss.2022.111302

4. Investigate some other proof assistants, such as Coq, Isabelle/HOL, PVS, and ACL2.

Exercises

5. Investigate automated theorem provers, such as E, Vampire, and Otter.

6. Investigate SMT solvers, such as Z3 and Yices.

7. Investigate any other tools that support/automate theorem proving.

8. Investigate the proof of the four-color problem conducted with Coq.

https://doi.org/10.1007/978-3-540-87827-8_28

https://www.ams.org/notices/200811/tx081101382p.pdf

https://github.com/coq-community/fourcolor