Computer-certified Proofs

Module: MATH5004M
Student ID: 200684195
Supervisor: Dr Nicola Gambino

Ismail Suleman

Overview of Coq

What is Coq

- Functional programming language
- Dependent types
- Construct certified programs, prove theorems

Type hierarchy

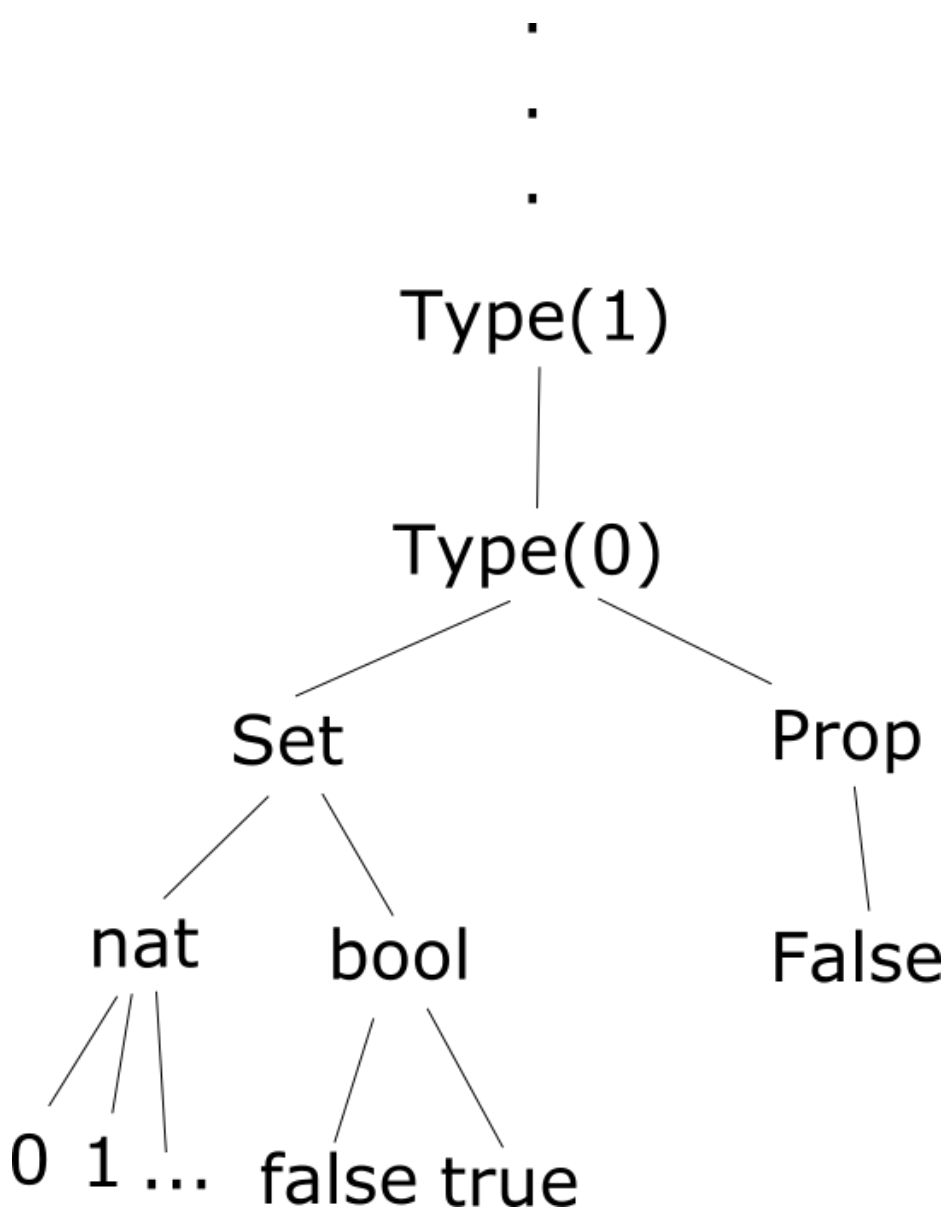

Figure 1: Hierarchy of types in Coq

Function types

- $\texttt{nat} \to \texttt{bool} \to \texttt{nat} \to \texttt{nat}$
- $\texttt{nat} \to (\texttt{bool} \to (\texttt{nat} \to \texttt{nat}))$

Dependent product

$$\frac{\texttt{A: s} \qquad \texttt{a: A} \vdash \texttt{B: s'}}{\texttt{forall a: A, B: s''}}$$

$$\frac{\texttt{A: Set} \qquad \texttt{a: A} \vdash \texttt{B: Prop}}{\texttt{forall a: A, B: Prop}}$$

Example code

```
(* Some example code *)
Section example_code.
  Variables (A: Set) (P Q: A → Prop) (R: A → A → Prop).

  Theorem all_perm: (forall a b: A, R a b) → forall a b: A, R b a.
  Proof (fun H a b ⇒ H b a).

  Theorem all_perm': (forall a b: A, R a b) → forall a b: A, R b a.
  Proof.
    intros H a b.
    apply H.
  Qed.
```

```
  Inductive Z_inf_branch_tree: Set :=
2   Z_inf_leaf: Z_inf_branch_tree
  | Z_inf_node: Z → (nat → Z_inf_branch_tree) →
    Z_inf_branch_tree.
```

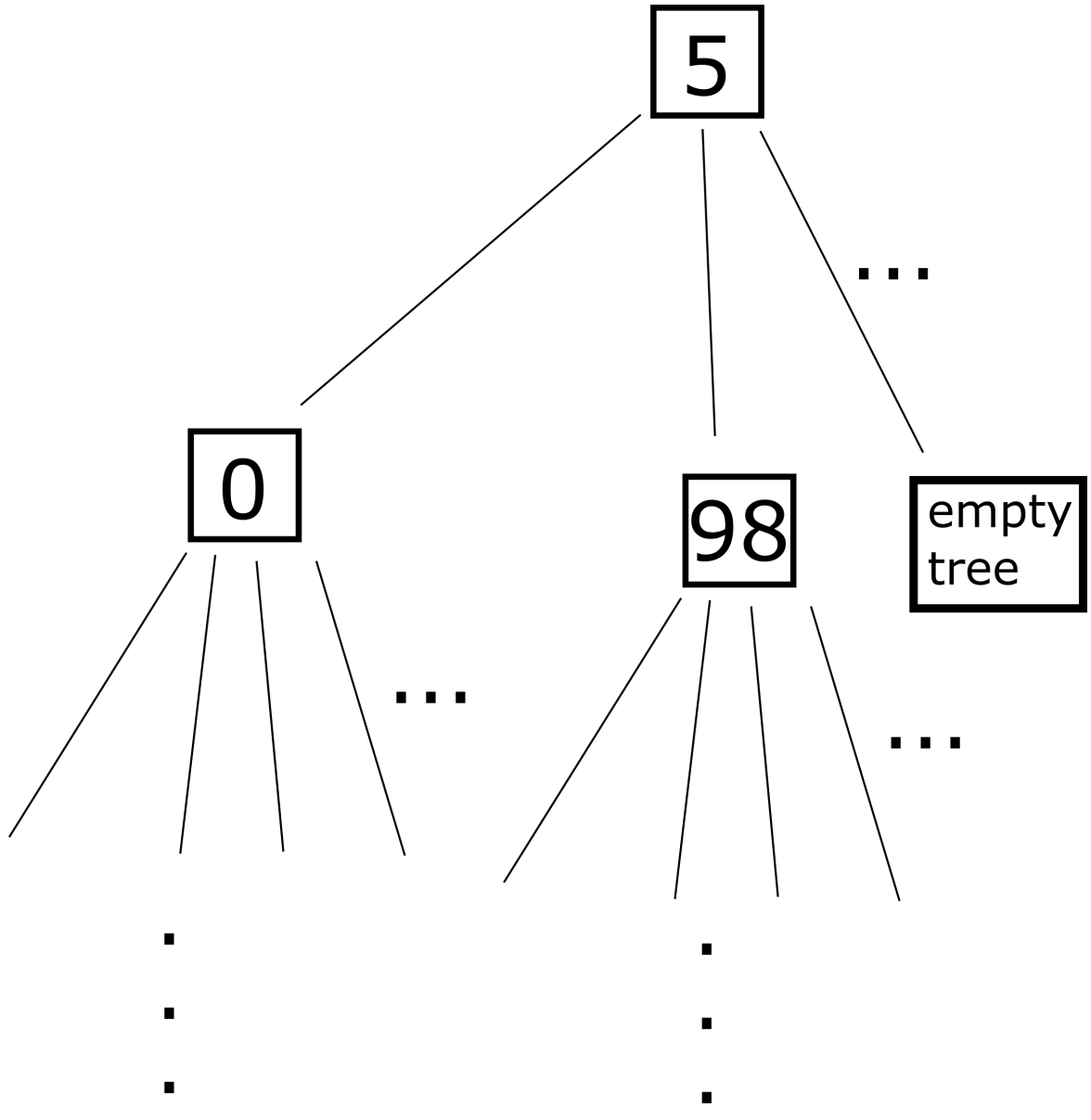

Figure 2: Graphical representation of `Z_inf_branch_tree`

```
  Fixpoint zero_in_tree (n: nat) (tree: Z_inf_branch_tree):
    bool :=
    match tree with
    | Z_inf_leaf ⇒ false
    | Z_inf_node 0%Z _ ⇒ true
    | Z_inf_node _ f ⇒ match n with
                       | 0 ⇒ zero_in_tree n (f 0)
                       | _ ⇒ (fix g (n': nat): bool :=
                                match n' with
                                | 0 ⇒ zero_in_tree n (f 0)
                                | S n'' ⇒ orb (zero_in_tree n
    (f n')) (g n'')
                                end ) n
                     end
    end.
End example_code.
```

Rewriting a subset of the standard library

Existing state of the standard library

- >40 folders, >400 files
- Code spread across files
- Large dependency tree
- Comments

```
Inductive between k : nat → Prop :=
  | bet_emp : between k k
  | bet_S : forall l, between k l → P l → between k (S l).
```

Rewriting a subset of the standard library

```
Theorem plus_comm : forall n m : nat, n + m = m + n. Admitted.
Theorem plus_assoc : forall n m p : nat, n + (m + p) = (n + m) +
    p. Admitted.

Theorem plus_swap : forall n m p : nat, n + (m + p) = m + (n +
    p).
Proof.
  intros n m p.
  assert (H: n + (m + p) = n + m + p).
  {rewrite → plus_assoc. reflexivity. }
  assert (H1: n + m = m + n).
  {rewrite → plus_comm. reflexivity. }
  rewrite → H. rewrite → H1. rewrite → plus_assoc.
    reflexivity.
Qed.
```

```
1 subgoal
n, m, p : nat
H : n + (m + p) = n + m + p
H1 : n + m = m + n
______________________________________(1/1)
n + (m + p) = m + (n + p)
```

```
rewrite → H.
```

```
1 subgoal
n, m, p : nat
H : n + (m + p) = n + m + p
H1 : n + m = m + n
______________________________________(1/1)
n + m + p = m + (n + p)
```

```
rewrite → H1.
```

```
1 subgoal
n, m, p : nat
H : n + (m + p) = n + m + p
H1 : n + m = m + n
______________________________________(1/1)
m + n + p = m + (n + p)
```

```
rewrite → plus_assoc.
```

```
1 subgoal
n, m, p : nat
H : n + (m + p) = n + m + p
H1 : n + m = m + n
______________________________________(1/1)
m + n + p = m + n + p
```

```
reflexivity. Qed.
```

Conclusion

- Coq is complicated, but powerful
- More rigorous proofs, but often less-readable