Introduction to Coq Programming

Chung-Kil Hur (허충길)

Department of Computer Science and Engineering
Seoul National University

2017 SIGPL 겨울 학교

진행 방식

수업: 이론 + 질문답변 + 데모

실습: 연습 문제 풀기

What is Coq?

Coq = Calculus of Construction

~ A functional programming language

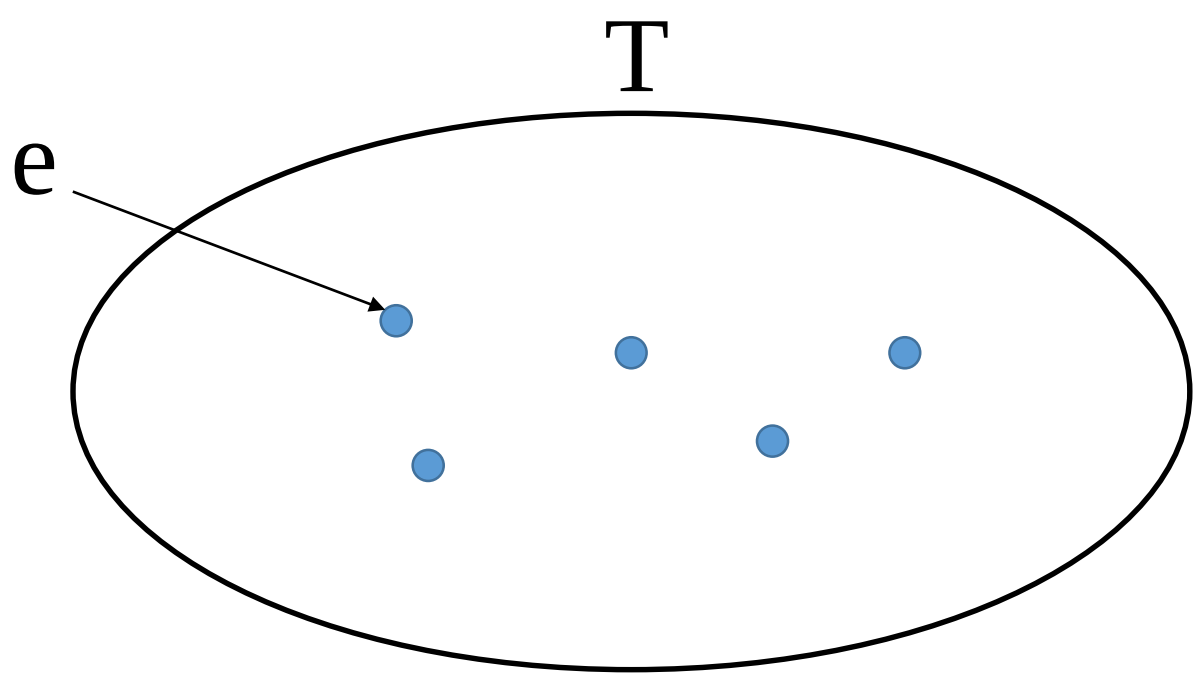
~ A foundation for mathematics

Functional Programming

Programming: Types and Expressions

expression Type

$$e : T$$



e.g. 3+4 : nat

7 : nat

Evaluation: Computation → Value

expression Type

$$e : T$$

expression = computation + value

e.g. 3+4 : nat (computation)

7 : nat (value)

Evaluation: computation → value

e.g. 3+4 → 7 : nat

Inductive & Function Types

	Type	Value	Computation
Inductive			
Function			

Inductive Types: Simple Case

	Type	Value	Computation
Inductive	Inductive T : Type := | c1 : T | c2 : T … .	c1 : T, c2 : T, …	match (e:T) with | c1 => (e1:S) | c2 => (e2:S) … end : S
Function			

Evaluation:

match c2 with
| c1 => e1
| c2 => e2 → e2
…
end

Function Types: Simple Case

	Type	Value	Computation
Inductive	Inductive T : Type := | c1 : T | c2 : T … .	c1 : T, c2 : T, …	match (e:T) with | c1 => (e1:S) | c2 => (e2:S) … end : S
Function	T -> S	(fun (x:T) => (e:S)) : T -> S	(e:T->S) (e1: T) : S

Evaluation:

(fun x => e) v1 → e[x ↦ v1]

* Names and Substitution play important roles

Inductive Types: General Case (Recursion)

	Type	Value	Computation
Inductive	Inductive T : Type := | c1 (x: S1) : T | c2 (x: S2) : T … .	c1(v:S1) : T, c2(v:S2) : T, …	match (e:T) with | c1(x) => (e1:S) | c2(x) => (e2:S) … end : S
Function	T -> S	(fun (x:T) => (e:S)) : T -> S	(e:T->S) (e1: T) : S

Evaluation:

match c*2*(v) with
| c1(x) => e1
| c2(x) => e2 → e*2*[x ↦ v]
…
end

Function Types: General Case (Recursion)

	Type	Value	Computation
Inductive	Inductive T : Type := | c1 (x: S1) : T | c2 (x: S2) : T … .	c1(v:S1) : T, c2(v:S2) : T, …	match (e:T) with | c1(x) => (e1:S) | c2(x) => (e2:S) … end : S
Function	T -> S	(fix (f:T->S) (x:T) := (e:S)) : T -> S	(e:T->S) (e1: T) : S

Evaluation:

(fix f x := e) e1 → e[f ↦ (fix f x := e)][x ↦ e1]

* Names and Substitution play important roles

Type: The type of all types

Type

For any type T, T : Type

Type : Type

Function Types: General Case (Dependent)

	Type	Value	Computation
Inductive	Inductive T : Type := | c1 (x: S1) : T | c2 (x: S2) : T … .	c1(v:S1) : T, c2(v:S2) : T, …	match (e:T) with | c1(x) => (e1:S) | c2(x) => (e2:S) … end : S
Function	forall (x:T), S	(fix (f:T->S) (x:T) := (e:S)) : T -> S	(e:T->S) (e1: T) : S

Evaluation:

(fix f x := e) v1 → e[f ↦ (fix f x := e)][x ↦ v1]

* Names and Substitution play important roles