Chapter 29: Type Operators and Kinding

Type-level Functions

Kinding

λω

Type Constructor and Type Function

- Type Constructor

$$\text{Pair } Y\ Z = \forall X.\ (Y \to Z \to X) \to X;$$

- Type Function: A map from Types to Types

$$\text{Pair} = \lambda Y.\ \lambda Z.\ \forall X.\ (Y \to Z \to X) \to X;$$

Type Constructor and Type Function

- Introducing abstraction and application at the level of types gives us the possibility of writing the same type in different ways.

$$Id = \lambda X.X$$

Then, the following are all equivalent:

$Nat \rightarrow Bool$

$Nat \rightarrow Id\ Bool$

$Id\ Nat \rightarrow Id\ Bool$

$Id\ (Nat \rightarrow Bool)$

$$(\lambda X{::}K_{11}.T_{12})\ T2 \equiv [X \rightarrow T_2]T_{12}$$

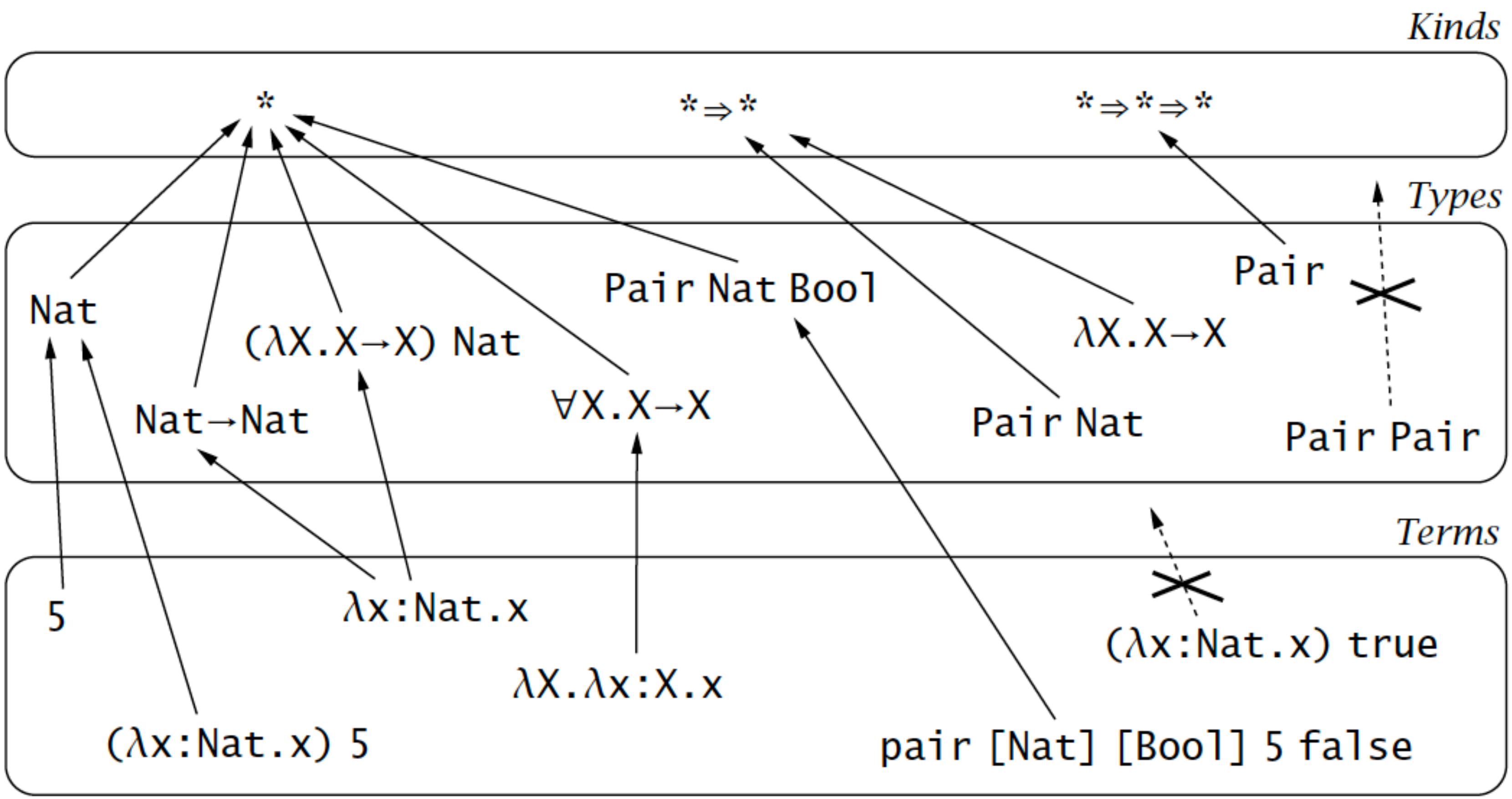
Kind

- Kinds, "the types of types", are introduced to avoid writing meaningless type expressions such as (Bool Nat)

$*$	the kind of proper types (like Bool and Bool→Bool)
$*\Rightarrow*$	the kind of type operators (i.e., functions from proper types to proper types)
$*\Rightarrow*\Rightarrow*$	the kind of functions from proper types to type operators (i.e., two-argument operators)
$(*\Rightarrow*)\Rightarrow*$	the kind of functions from type operators to proper types

Pair = λA::*. λB::*. ∀ X. (A→B→X) → X;

Terms, Types, and Kinds



Proper types: types with kind *

Type operators and kinding (λω)

Syntax

`t`	::=		*terms:*
		`x`	*variable*
		`λx:T.t`	*abstraction*
		`t t`	*application*
`v`	::=		*values:*
		`λx:T.t`	*abstraction value*

Type operators and kinding (λω)

T	::=		*types:*
		X	*type variable*
		$\lambda X :: K . T$	*operator abstraction*
		T T	*operator application*
		$T \rightarrow T$	*type of functions*
Γ	::=		*contexts:*
		∅	*empty context*
		Γ, x : T	*term variable binding*
		Γ, X :: K	*type variable binding*
K	::=		*kinds:*
		*	*kind of proper types*
		$K \Rightarrow K$	*kind of operators*

Type operators and kinding (λω)

Evaluation $\boxed{t \longrightarrow t'}$

$$\frac{t_1 \longrightarrow t_1'}{t_1\ t_2 \longrightarrow t_1'\ t_2} \quad \text{(E-App1)}$$

$$\frac{t_2 \longrightarrow t_2'}{v_1\ t_2 \longrightarrow v_1\ t_2'} \quad \text{(E-App2)}$$

$$(\lambda x{:}T_{11}.t_{12})\ v_2 \longrightarrow [x \mapsto v_2]t_{12} \quad \text{(E-AppAbs)}$$

Type operators and kinding (λω)

Kinding $\boxed{\Gamma \vdash T :: K}$

$$\frac{X :: K \in \Gamma}{\Gamma \vdash X :: K} \quad \text{(K-TVar)}$$

$$\frac{\Gamma, X :: K_1 \vdash T_2 :: K_2}{\Gamma \vdash \lambda X :: K_1 . T_2 :: K_1 \Rightarrow K_2} \quad \text{(K-Abs)}$$

$$\frac{\Gamma \vdash T_1 :: K_{11} \Rightarrow K_{12} \qquad \Gamma \vdash T_2 :: K_{11}}{\Gamma \vdash T_1\ T_2 :: K_{12}} \quad \text{(K-App)}$$

$$\frac{\Gamma \vdash T_1 :: * \qquad \Gamma \vdash T_2 :: *}{\Gamma \vdash T_1 \rightarrow T_2 :: *} \quad \text{(K-Arrow)}$$

Type operators and kinding (λω)

Type equivalence $\boxed{S \equiv T}$

$$T \equiv T \qquad \text{(Q-Refl)}$$

$$\frac{T \equiv S}{S \equiv T} \qquad \text{(Q-Symm)}$$

$$\frac{S \equiv U \qquad U \equiv T}{S \equiv T} \qquad \text{(Q-Trans)}$$

$$\frac{S_1 \equiv T_1 \qquad S_2 \equiv T_2}{S_1 \to S_2 \equiv T_1 \to T_2} \qquad \text{(Q-Arrow)}$$

$$\frac{S_2 \equiv T_2}{\lambda X{::}K_1 . S_2 \equiv \lambda X{::}K_1 . T_2} \qquad \text{(Q-Abs)}$$

$$\frac{S_1 \equiv T_1 \qquad S_2 \equiv T_2}{S_1\ S_2 \equiv T_1\ T_2} \qquad \text{(Q-App)}$$

$$(\lambda X{::}K_{11} . T_{12})\ T_2 \equiv [X \mapsto T_2]T_{12} \qquad \text{(Q-AppAbs)}$$

Type operators and kinding (λω)

Typing $\boxed{\Gamma \vdash t : T}$

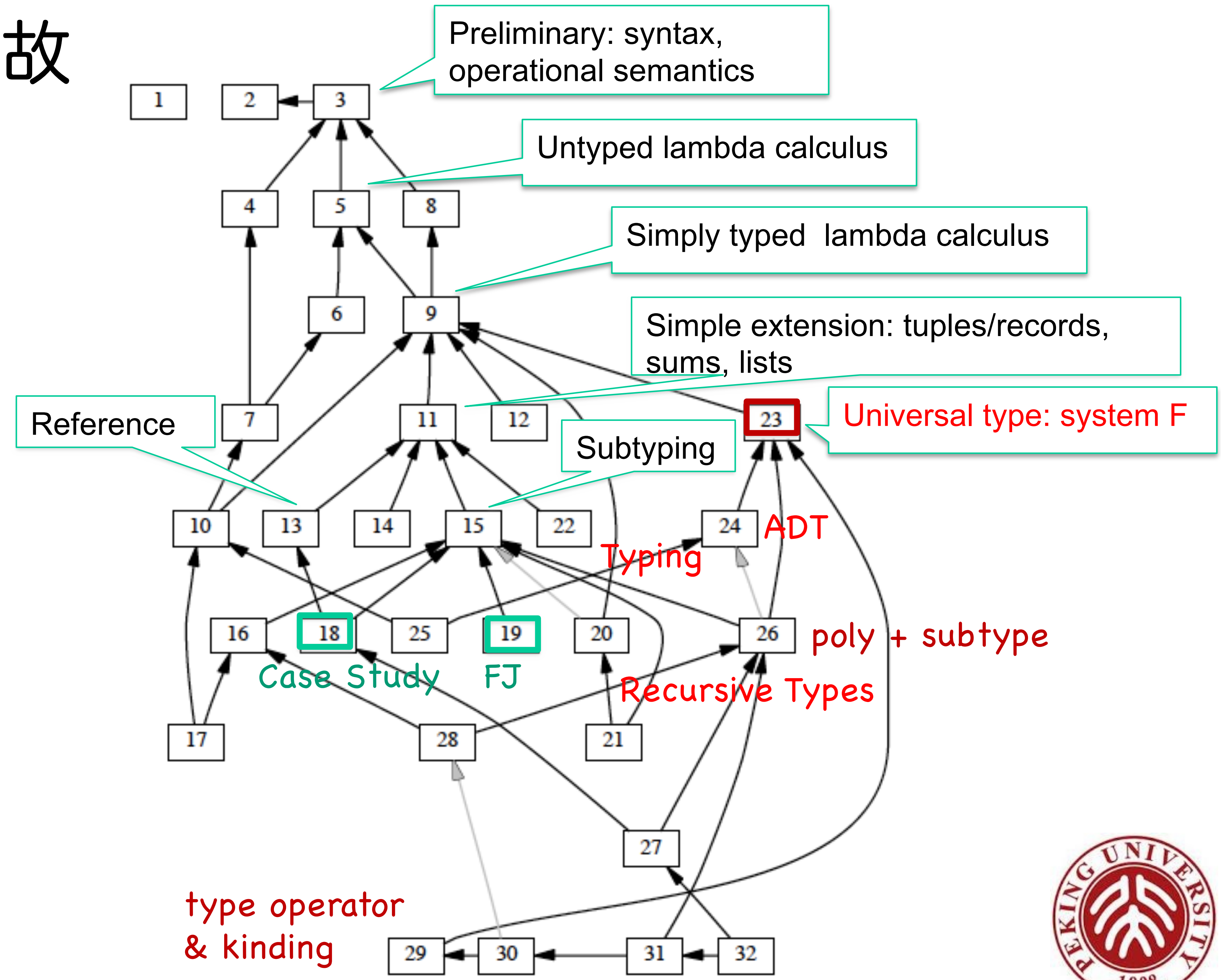
$$\frac{x{:}T \in \Gamma}{\Gamma \vdash x : T} \qquad \text{(T-VAR)}$$

$$\frac{\Gamma \vdash T_1 :: * \qquad \Gamma, x{:}T_1 \vdash t_2 : T_2}{\Gamma \vdash \lambda x{:}T_1.t_2 : T_1 \to T_2} \qquad \text{(T-ABS)}$$

$$\frac{\Gamma \vdash t_1 : T_{11} \to T_{12} \qquad \Gamma \vdash t_2 : T_{11}}{\Gamma \vdash t_1\ t_2 : T_{12}} \qquad \text{(T-APP)}$$

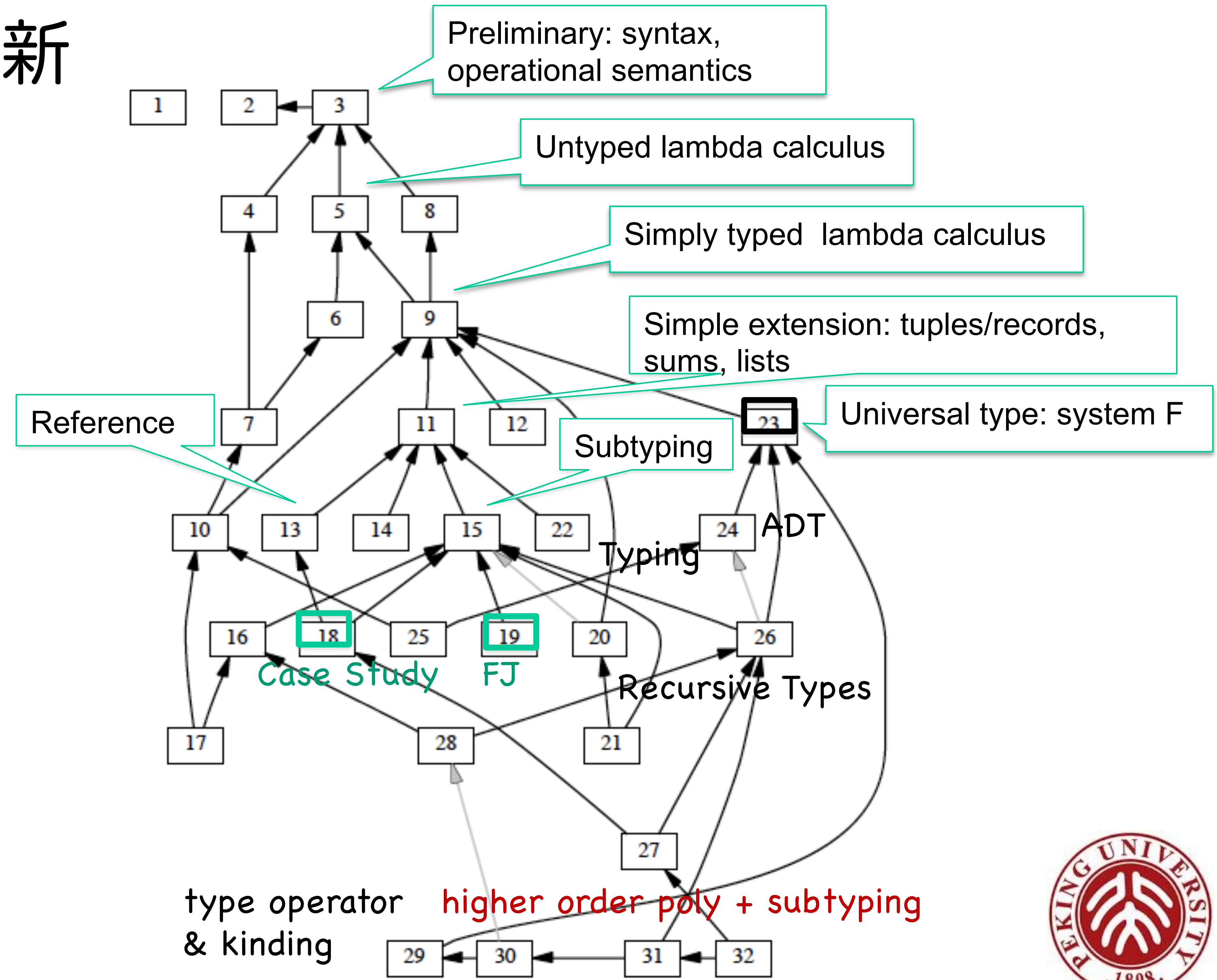
$$\frac{\Gamma \vdash t : S \qquad S \equiv T \qquad \Gamma \vdash T :: *}{\Gamma \vdash t : T} \qquad \text{(T-EQ)}$$

温故



Preliminary: syntax, operational semantics
Untyped lambda calculus
Simply typed lambda calculus
Simple extension: tuples/records, sums, lists
Reference
Subtyping
Universal type: system F
ADT
Typing
poly + subtype
Case Study
FJ
Recursive Types
type operator & kinding
1
2
3
4
5
8
6
9
7
11
12
23
10
13
14
15
22
24
16
18
25
19
20
26
17
28
21
27
29
30
31
32

知新


Preliminary: syntax, operational semantics
Untyped lambda calculus
Simply typed lambda calculus
Simple extension: tuples/records, sums, lists
Universal type: system F
Reference
Subtyping
ADT
Typing
Case Study
FJ
Recursive Types
type operator & kinding
higher order poly + subtyping
1
2
3
4
5
8
6
9
7
11
12
23
10
13
14
15
22
24
16
18
25
19
20
26
17
28
21
27
29
30
31
32

谢谢大家的课堂配合！
欢迎与北大程序设计语言实验室合作！