

# **Foundations of Certified Programming Language and Compiler Design**

Dr.-Ing. Sebastian Ertel

Composable Operating Systems Group, Barkhausen Institute

## Outline



ecture	Logic Propositional and first-order logic	Formalisms	PL
2	Tropositional and mot order logic		Functional programming
3		Syntax and Semantics	
4			The untyped lambda calculus
5		Types	
6			The typed lambda calculus
7			Polymorphism
8		Curry-Howard	
9			Higher-order types
10			Dependent types

# The simply typed lambda calculus



#### Syntax:

#### Semantics:

# The simply typed lambda calculus



#### Syntax:

## **Typing**

$$\Gamma \vdash t : T$$

$$\frac{x:T\in\Gamma}{\Gamma\vdash x:T} \text{ T-Var }$$

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

$$\frac{\Gamma \vdash t_1: T_{11} \rightarrow T_{12} \quad \Gamma \vdash t_2: T_{11}}{\Gamma \vdash t_1 \ t_2: T_{12}} \quad \text{T-App}$$

#### **Derivation Trees**



Typing derviations:

$$\frac{x: \texttt{Bool} \in x: \texttt{Bool}}{x: \texttt{Bool} \vdash x: \texttt{Bool}} \xrightarrow{\texttt{T-VAR}} \frac{x: \texttt{Bool} \vdash x: \texttt{Bool}}{\texttt{Ax}: \texttt{Bool}.x: \texttt{Bool} \to \texttt{Bool}} \xrightarrow{\texttt{T-ABS}} \frac{\texttt{T-TRUE}}{\vdash \texttt{true}: \texttt{Bool}} \xrightarrow{\texttt{T-APP}}$$

Check: Show the derivation tree for

 $f: \mathtt{Bool} o \mathtt{Bool} \vdash f \ (\mathtt{if} \ \mathtt{false} \ \mathtt{then} \ \mathtt{true} \ \mathtt{else} \ \mathtt{false}) : \mathtt{Bool}$ 

$$\frac{f: \texttt{Bool} \to \texttt{Bool} \in \Gamma}{\Gamma \vdash f: \texttt{Bool} \to \texttt{Bool}} \xrightarrow{\mathsf{T-VAR}} \frac{\overline{\mathsf{false} : \texttt{Bool}}}{\Gamma \vdash \mathsf{if}} \xrightarrow{\mathsf{false} : \texttt{Bool}} \frac{\mathsf{T-FALSE}}{\mathsf{true} : \texttt{Bool}} \xrightarrow{\mathsf{T-TRUE}} \frac{\overline{\mathsf{false} : \texttt{Bool}}}{\mathsf{T-IF}} \xrightarrow{\mathsf{T-APP}} \frac{\mathsf{T-FALSE}}{\mathsf{T-IF}}$$

• Check: Find a context  $\Gamma$  for  $f \ x \ y$  : Bool.

```
\Gamma = f: \mathtt{Bool} 	o \mathtt{Bool} 	o \mathtt{Bool}, \qquad f: \mathtt{Nat} 	o \mathtt{Nat} 	o \mathtt{Bool}, \qquad f: \mathtt{T} 	o \mathtt{T} 	o \mathtt{Bool} x: \mathtt{Bool}, y: \mathtt{Bool} x: \mathtt{Nat}, y: \mathtt{Nat} x: \mathtt{T}, y: \mathtt{T}
```

## **Properties**



#### Lemmas:

Inversion of the typing relation.

Canonical forms.

#### Theorems:

Uniqueness of types.

## Theorem (Progress)

Suppose t is a closed, well-typed term, i.e.,  $\vdash t: T$ . Then either t is a value or there exists some t' such that  $t \longrightarrow t'$ .

## Theorem (Normalization)

If  $\vdash t : T$ , then t is normalizable.

## Theorem (Preservation)

If  $\Gamma \vdash t : T$  and  $t \longrightarrow t'$ , then  $\Gamma \vdash t' : T$ 

What is the type of  $(\lambda x.x \ x) \ (\lambda x.x \ x)$ ?

## Extensions



- Currently, we cannot implement the STLC because we are missing the base case for our types.
- Let's study extensions:

# Extensions Base Types



- Base types: Nat, Bool, String, Float, ...
- Let's add some uninterpreted/unkown base types without any primitive operations.
   New syntactic forms:

$$T ::= \dots$$
 types:   
 | A base type

- Consider:  $\lambda x : A. \ x : A \rightarrow A$
- We could assume that  $A=\mathtt{Nat}$  is some number:  $(\lambda x:A.\ x:A\to A)\ 5$

#### **Extensions**



- But where do these values come from and what actually is Nat?
- We do not want to add unknown values/types?!
- We would like to have a closed system to reason about it.

## Unit



## New syntactic forms:

## New Typing Rules:

```
\boxed{\Gamma \vdash t : T} \boxed{\frac{}{\text{unit} : \text{Unit}}} \ \text{T-Unit}
```

# Ascription



Document your code ... with types!
 New syntactic forms:

```
t ::= \dots terms:
```

New Evaluation Rules:

$$t \longrightarrow t'$$

$$\overline{v_1 \text{ as } T_1 \longrightarrow v_1}$$
 E-Ascribe1

$$rac{t_1 \longrightarrow t_1'}{t_1 ext{ as } T_1 \longrightarrow t_1' ext{ as } T_1} ext{ E-Ascribe2}$$

**New Typing Rules:** 

$$\Gamma \vdash t : T$$

$$rac{\Gamma dash t_1:T_1}{\Gamma dash t_1 ext{ as } T_1:T_1}$$
 T-Ascribe

# Product Types a.k.a. Pairs a.k.a. Tuples (with two elements)



## New syntactic forms:

#### New Evaluation Rules:

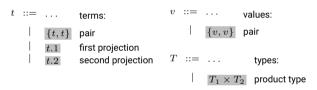
$$t \longrightarrow t'$$

$$\frac{t_1 \longrightarrow t_1'}{\{v_1,v_2\}.1 \longrightarrow v_1} \text{ E-PairBeta1} \qquad \frac{t_1 \longrightarrow t_1'}{\{v_1,v_2\}.2 \longrightarrow v_2} \text{ E-PairBeta2} \qquad \frac{t_1 \longrightarrow t_1'}{t_1.1 \longrightarrow t_1'.1} \text{ E-ProJ1}$$
 
$$\frac{t_2 \longrightarrow t_2'}{t_2.2 \longrightarrow t_2'.2} \text{ E-ProJ2} \qquad \frac{t_1 \longrightarrow t_1'}{\{t_1,t_2\} \longrightarrow \{t_1',t_2\}} \text{ E-Pair1} \qquad \frac{t_2 \longrightarrow t_2'}{\{v_1,t_2\} \longrightarrow \{v_1,t_2'\}} \text{ E-Pair2}$$

# Product Types a.k.a. Pairs a.k.a. Tuples (with two elements)



## New syntactic forms:



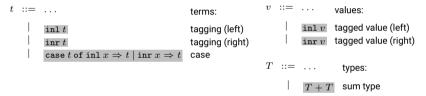
## New Typing Rules:

$$\frac{\Gamma \vdash t:T}{\Gamma \vdash \{t_1,t_2\}:T_1\times T_2} \quad \text{T-Pair} \qquad \frac{\Gamma \vdash t:T_1\times T_2}{\Gamma \vdash t.1:T_1} \quad \text{T-ProJ1} \qquad \frac{\Gamma \vdash t:T_1\times T_2}{\Gamma \vdash t.2:T_2} \quad \text{T-ProJ2}$$

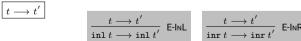
# Sum Types



## New syntactic forms:



#### New Evaluation Rules:



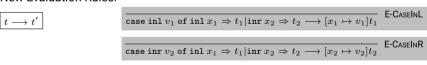
## Sum Types



#### New syntactic forms:



#### **New Evaluation Rules:**



$$\frac{t_0 \longrightarrow t_0'}{\operatorname{case} t_0 \text{ of inl } x_1 \Rightarrow t_1 | \operatorname{inr} x_2 \Rightarrow t_2 \longrightarrow \operatorname{case} t_0' \text{ of inl } x_1 \Rightarrow t_1 | \operatorname{inr} x_2 \Rightarrow t_2} \text{ E-CASE}$$

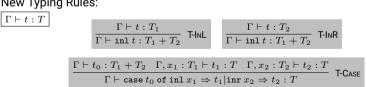
# Sum Types



#### New syntactic forms:

```
t ::= ...
                                                                                        values:
                                                 terms:
                                                 tagging (left)
                                                                                       tagged value (left)
                                                                                        tagged value (right)
                                                 tagging (right)
         case t of inl x \Rightarrow t \mid \text{inr } x \Rightarrow t
                                                 case
                                                                     T ::= ...
                                                                                          types:
                                                                                          sum type
```

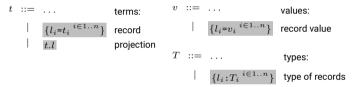
## New Typing Rules:



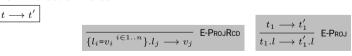
## Labeled Product Types a.k.a. Records



## New syntactic forms:



#### **New Evaluation Rules:**



## Labeled Product Types a.k.a. Records



#### New syntactic forms:

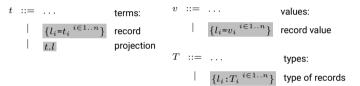
#### **New Evaluation Rules:**

$$\frac{t\longrightarrow t'}{\{l_i=v_i\ ^{i\in 1...j-1},l_j=t_j,l_k=t_k\ ^{k\in j+1...n}\}} \ \to \{l_i=v_i\ ^{i\in 1...j-1},l_j=t'_i,l_k=t_k\ ^{k\in j+1...n}\}} \ \ \text{E-Rcd}$$

# Labeled Product Types a.k.a. Records



## New syntactic forms:



## New Typing Rules:

$$\Gamma \vdash t : T$$

$$\frac{\text{for each } i \quad \Gamma \vdash t_i : T_i}{\Gamma \vdash \{l_i = t_i \ ^{i \in 1 \dots n}\} : \{l_i : T_i \ ^{i \in 1 \dots n}\}} \text{ T-RcD}$$

$$\frac{\Gamma \vdash t_1:\{l_i{:}T_i^{\ i\in 1..j-1},l_j{:}T_j,l_k{:}T_k^{\ k\in j+1..n}\}}{\Gamma \vdash \{t_1.l_j\}:T_j} \ \text{ T-Proj}$$

# Sum Types (again) Uniqueness of Typing



 $\label{lem:valid_types} \mbox{ for inl 5 are Nat} + \mbox{Nat} + \mbox{Bool etc.}$ 

## New syntactic forms:

#### **New Evaluation Rules:**

```
\frac{t \longrightarrow t'}{\mathsf{case}\,(\mathsf{inl}\,v_1 \,\mathsf{las}\,T)\mathsf{of}\,\mathsf{inl}\,x_1 \Rightarrow t_1|\mathsf{inr}\,x_2 \Rightarrow t_2 \longrightarrow [x_1 \mapsto v_1]t_1} \quad \mathsf{E\text{-}CaseINL}} \frac{\mathsf{case}\,(\mathsf{inr}\,v_2 \,\mathsf{las}\,T)\mathsf{of}\,\mathsf{inl}\,x_1 \Rightarrow t_1|\mathsf{inr}\,x_2 \Rightarrow t_2 \longrightarrow [x_2 \mapsto v_2]t_2} \quad \mathsf{E\text{-}CaseINR}}
```

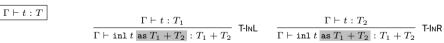
# Sum Types (again) Uniqueness of Typing



Valid types for inl 5 are Nat + Nat, Nat + Bool etc.

## New syntactic forms:

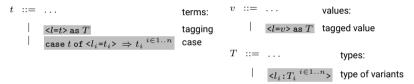
## New Typing Rules:



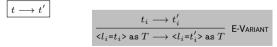
## Labeled Sum Types a.k.a. Variants a.k.a. Algebraic Datatypes



#### New syntactic forms:



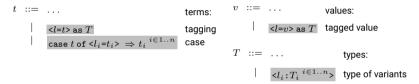
#### New Evaluation Rules:



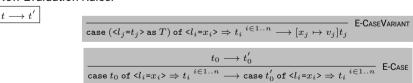
## Labeled Sum Types a.k.a. Variants a.k.a. Algebraic Datatypes



#### New syntactic forms:



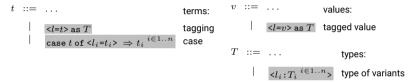
#### **New Evaluation Rules:**



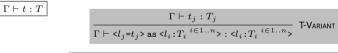
## Labeled Sum Types a.k.a. Variants a.k.a. Algebraic Datatypes



#### New syntactic forms:



## **New Typing Rules:**



$$\frac{\Gamma \vdash t_0 : <\!\! l_i \!:\! T_i \stackrel{i \in 1 \ldots n}{>} \quad \text{for each } i \quad \Gamma, x_i : T_i \vdash t_i : T}{\Gamma \vdash \mathsf{case} \ t_0 \ \mathsf{of} <\!\! l_i \!=\!\! x_i \!>} \Rightarrow t_i \stackrel{i \in 1 \ldots n}{>} : T$$
 T-CASE

## **Examples**



#### STLC++

#### Coq

```
Inductive bool := False | True.

Inductive week = Weekday | Weekend.

Inductive optionNat = None | Some (_:nat_
Inductive natList = Nil | Cons (_:nat) (_:natList).
Inductive nat = Zero | Succ (:nat).
```