UNIVERSITY OF Southampton
School of Electronics
and Computer Science

COMP2212
PROGRAMMING LANGUAGE CONCEPTS

Dr Julian Rathke

ADDING FUNCTIONS TO TOY

HOW ABOUT ADDING FUNCTIONS TO TOY?

- We could consider how to add **Lambda Calculus** like functions to our Toy language.
- We would need to add function abstraction and application as operations.
- We introduced Lambda Calculus in COMP2209 where we looked at it as an untyped language.
- Recall that, in its untyped form, it is Turing complete - that is, all computation can be expressed in it !
- This situation changes somewhat if we introduce simple types.

A note on notation : the exact syntax for lambda calculus varies from source to source so sometimes you will see $\lambda\, x \to E$, sometimes $\lambda\, x\,.\, E$ and sometimes $\lambda\,(x)\, E$ - they all mean the same thing.

SOME EXAMPLE LAMBDA CALCULUS TERMS

The identity function: $\lambda x . x$ is a function that takes a single argument and simply returns it.

First projection: $\lambda x . \lambda y . x$ is a function that takes two arguments and returns the first

Second projection: $\lambda x . \lambda y . y$ is a function that takes two arguments and returns the second

Twice : $\lambda f . \lambda x . f (f (x))$ is a function that takes a single-argument function and an argument and twice applies the supplied function f to the argument x.

Composition : $\lambda g . \lambda f . \lambda x . f (g (x))$ takes two functions and an argument and returns the composed function f ; g

Let's think about what types these expressions might have?

We need to allow some kind of type for functions e.g. T → U as a function that takes data of type T and returns data of type U

SIMPLY-TYPED LAMBDA CALCULUS

We can formalise a simple type system for lambda calculus. Without some base types though this is a very uninteresting language so let's look at it combined with the Toy language. We'll call the resulting language **λ**Toy

```
T , U ::=  Int | Bool | T → T
E ::=  n | true | false | E < E | E + E | x |
        | if E then E else E | λ (x : T) E |
        | let (x : T) = E in E | E E
```

The type rules for the added constructs are straightforward given what we have learnt already about type environments:

$$\frac{\Gamma, x : T \vdash E : U}{\Gamma \vdash \lambda(x : T)E : T \rightarrow U}\textsc{TLam}$$

$$\frac{\Gamma \vdash E_1 : T \rightarrow U \quad \Gamma \vdash E_2 : T}{\Gamma \vdash E_1\ E_2 : U}\textsc{TApp}$$

TYPES IN LAMBDA CALCULUS ?

Try using the typing rules from the previous slide and rule TVar to write type derivations of each of these lambda expressions.

The identity function: $\lambda\ (x{:}T)\ x \quad : T \to T$

First projection: $\lambda\ (x{:}T)\ \lambda\ (y : U)\ x \quad : T \to (U \to T)$

Second projection: $\lambda\ (x : T)\ \lambda\ (y : U)\ y \quad : T \to (U \to U)$

Twice : $\lambda\ (f : T \to T)\ \lambda\ (x : T)\ \ f\ (\ f\ (\ x\ )\ ) : (T \to T) \to T \to T$

Composition : $\lambda\ (g : T \to U)\ \lambda\ (f : U \to V)\ \lambda\ (x : T)\ \ f\ (\ g\ (\ x\ )\ ) : (T \to U) \to (U \to V) \to (T \to V)$

TYPES IN LAMBDA CALCULUS - DERIVATION 1

The identity function: $\lambda\ (x{:}T)\ x \ : T \to T$

$$\dfrac{\dfrac{x : T \in \{ x : T \}}{x : T \vdash x : T}\ \text{TVar}}{\vdash \lambda\ (x : T)\ x \ : T \to T}\ \text{TLam}$$

TYPES IN LAMBDA CALCULUS - DERIVATION 2

First projection: $\lambda\,(x{:}T)\,\lambda\,(y:U)\;x\;:\;T \to (U \to T)$

x is the bound variable

$$
\dfrac{\dfrac{\dfrac{x:T \in \{\,x:T\,,\,y:U\,\}}{x:T\,,\;y:U\;\vdash x\;:\;T}\;\text{TVar}}{x:T \vdash \lambda\,(\,y:U\,)\,x\;:\;U \to T}\;\text{TLam}}{\vdash\;\lambda\,(\,x:T\,)\,\lambda\,(\,y:U\,)\,x\;\;:\,T \to (\,U \to T\,)}\;\text{TLam}
$$

TYPES IN LAMBDA CALCULUS - DERIVATION 3

Second projection: $\lambda\,(x : T)\ \lambda\,(y : U)\ y\ : T \to U \to U$

y is the bound variable

$$\dfrac{\dfrac{\dfrac{y : U \in \{\, x : T,\ y : U \,\}}{x : T,\ y : U \vdash y : U}\ \text{TVar}}{x : T \vdash \lambda\,(y : U)\ y : U \to U}\ \text{TLam}}{\vdash \lambda\,(x : T)\ \lambda\,(y : U)\ y : T \to (U \to U)}\ \text{TLam}$$

TYPES IN LAMBDA CALCULUS - DERIVATION 4

Twice : $\lambda\,(f:T \to T)\ \lambda\,(x:T)\ \ f\,(\,f\,(\,x\,)\,)\ :\ (T \to T) \to T \to T$

$$
\dfrac{
  \dfrac{
    \dfrac{f:T \to T \in \{\, f:T \to T\,,\, x:T \,\}}{f:T \to T\,,\ x:T\ \vdash f:\ T \to T}\,\text{TVar}
    \qquad
    \dfrac{
      \dfrac{f:T \to T \in \{\, f:T \to T\,,\, x:T \,\}}{f:T \to T\,,\ x:T\ \vdash f:\ T \to T}\,\text{TVar}
      \qquad
      \dfrac{x:T \in \{\, f:T \to T\,,\, x:T \,\}}{f:T \to T\,,\ x:T\ \vdash x\ :\ T}\,\text{TVar}
    }{f:T \to T\,,\ x:T\ \vdash f\ x\ :\ T}\,\text{TApp}
  }{
    \dfrac{f:T \to T\,,\ x:T\ \vdash f\,(\,f\ x\,)\ :\ T}{f:T \to T \vdash \lambda\,(\,x:T\,)\ f\,(\,f\ x\,)\ :\ T \to T}\,\text{TLam}
  }\,\text{TApp}
}{\vdash\ \lambda\,(\,f:T \to T)\ \lambda\,(\,x:T\,)\ \ f\,(\,f\ x\,)\ :\ (T \to T) \to T \to T}\,\text{TLam}
$$

TYPES IN LAMBDA CALCULUS - DERIVATION 5

Composition : $\lambda\,(g : T \to U)\ \lambda\,(f : U \to V)\ \lambda\,(x : T)\ \ f\,(\,g\,(\,x\,)\,)\ :\ (T \to U) \to (U \to V) \to T \to V$

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{f : U \to V \in \{\, g : T \to U,\ f : U \to V,\ x : T \,\}}{g : T \to U,\ f : U \to V,\ x : T \vdash f : U \to V}\ \text{TVar} \qquad \dfrac{\dfrac{g : T \to U \in \{\, g : T \to U,\ f : U \to V,\ x : T \,\}}{g : T \to U,\ f : U \to V,\ x : T \vdash g : T \to U}\ \text{TVar} \qquad \dfrac{x : T \in \{\, g : T \to U,\ f : U \to V,\ x : T \,\}}{g : T \to U,\ f : U \to V,\ x : T \vdash x : T}\ \text{TVar}}{g : T \to U,\ f : U \to V,\ x : T \vdash g\ \ x : U}\ \text{TApp}}{g : T \to U,\ f : U \to V,\ x : T \vdash f\,(\,g\,(\,x\,)\,)\ : V}\ \text{TApp}}{g : T \to U,\ f : U \to V \vdash \lambda\,(x : T)\ \ f\,(\,g\,(\,x\,)\,)\ : T \to V}\ \text{TLam}}{g : T \to U \vdash \lambda\,(\,f : U \to V\,)\ \lambda\,(x : T)\ \ f\,(\,g\,(\,x\,)\,)\ :\ (U \to V) \to T \to V}\ \text{TLam}}{\vdash\ \lambda\,(\,g : T \to U)\ \lambda\,(\,f : U \to V\,)\ \lambda\,(x : T)\ \ f\,(\,g\,(\,x\,)\,)\ :\ (T \to U) \to (U \to V) \to T \to V}\ \text{TLam}$$

THE TYPE SYSTEM IN FULL (FOR REFERENCE)

$$\frac{}{\Gamma \vdash n : \mathsf{Int}}\ \textsc{TInt} \qquad \frac{}{\Gamma \vdash b : \mathsf{Bool}}\ \textsc{TBool} \qquad \frac{x : T \in \Gamma}{\Gamma \vdash x : T}\ \textsc{TVar}$$

$$\frac{\Gamma \vdash E_1 : \mathsf{Int} \quad \Gamma \vdash E_2 : \mathsf{Int}}{\Gamma \vdash E_1 < E_2 : \mathsf{Bool}}\ \textsc{TLt} \qquad \frac{\Gamma \vdash E_1 : \mathsf{Int} \quad \Gamma \vdash E_2 : \mathsf{Int}}{\Gamma \vdash E_1 + E_2 : \mathsf{Int}}\ \textsc{TAdd}$$

$$\frac{\Gamma \vdash E_b : \mathsf{Bool} \quad \Gamma \vdash E_1 : T \quad \Gamma \vdash E_2 : T}{\Gamma \vdash \mathsf{if}\ E_b\ \mathsf{then}\ E_1\ \mathsf{else}\ E_2 : T}\ \textsc{TIf}$$

Notice the Γ in the TInt, TBool and TIf rules.

$$\frac{\Gamma \vdash E_1 : T \quad \Gamma, x : T \vdash E_2 : U}{\Gamma \vdash \mathsf{let}\ (x : T)\ =\ E_1\ \mathsf{in}\ E_2 : U}\ \textsc{TLet}$$

$$\frac{\Gamma, x : T \vdash E : U}{\Gamma \vdash \lambda(x : T)E : T \to U}\ \textsc{TLam}$$

$$\frac{\Gamma \vdash E_1 : T \to U \quad \Gamma \vdash E_2 : T}{\Gamma \vdash E_1\ E_2 : U}\ \textsc{TApp}$$

Phew, I'm glad that real programming languages don't have more constructs than **λ** Toy!

NEXT LECTURE: TYPE CHECKING