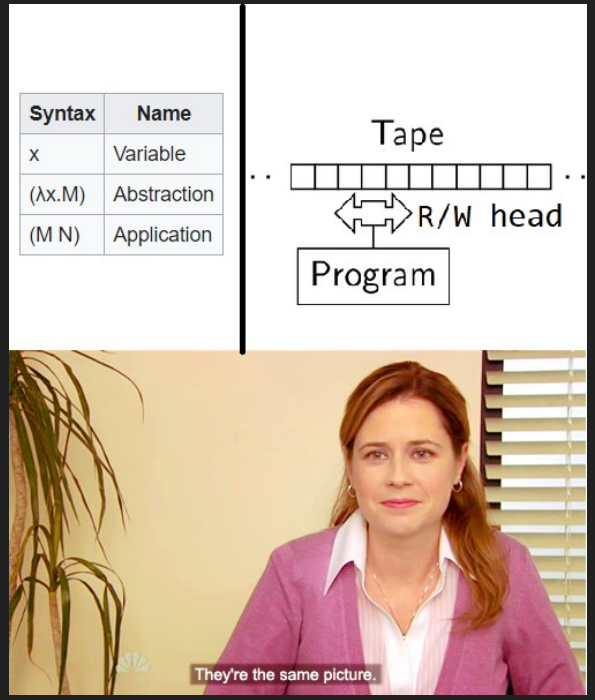
FIT2102 Programming Paradigms Tutorial 5

Lambda Calculus and Combinators

Syntax	Name
x	Variable
(λx.M)	Abstraction
(M N)	Application



Faculty of Information Technology

Housekeeping

- Cameras on for interviews
- Get your code ready before the interview
- Observables due this week (together with this week’s tute) - marked next week

Lambda Calculus

$I = \lambda x . x$ lambda calculus expression $(\lambda x . x) y$

```
I = x => x
```
JavaScript
```
(x => x)(y)
```

Alpha Equivalence

- expressions are equivalent if their variables are renamed
- $\lambda x\, y\, .\, x\, z = \lambda a\, b\, .\, a\, c$

Beta Reduction

- Application of functions involves substituting the argument into the expression

$$
\begin{aligned}
&= (\lambda\, x\, y\, .\, x)\ (\lambda\, x\, .\, x) \\
&= (\lambda\, x\, [x := \lambda\, x\, .\, x]\ y\ .\ x) \quad \Leftarrow \text{Beta reduction} \\
&= \lambda\, y\, .\ \ (\lambda\, x\, .\, x) \\
&= \lambda\, yx\, .\, x \quad \Leftarrow \text{Equivalent due to currying} \\
&= \lambda\, xy\, .\, y \quad \Leftarrow \text{Alpha equivalence}
\end{aligned}
$$

– The same thing in javascript!

```
> const f = (x => y => x)(x => x)
<- undefined
> f(2)(3)
<- 3
```

Eta Conversion

- Wrapping a simple lambda around an expression does not change the expression

$$\lambda x . M x = M$$

⇧

Eta conversion

Combinators

Combinators are functions which are expressions of only their parameters

They let us combine and transform other functions in various ways

```
const
   I = x=> x
 ,
   K = x=> y=> x
```

Identity: more useful than you might think!
e.g. test map: a.map(I) == a

Ignore the second parameter.
Where have we seen this before?