Programming Paradigms

Lab 1. Programming in Lambda Calculus

Outline

- Lambda calculus recap
- Programming in lambda calculus
  - Church Booleans
  - Church Pairs
  - Church Numerals
  - Enriching the calculus
  - Recursion

Lambda calculus recap

$\rightarrow$ *(untyped)*

Syntax		
$\mathtt{t}$ ::=		*terms:*
	$\mathtt{x}$	*variable*
	$\lambda\mathtt{x.t}$	*abstraction*
	$\mathtt{t\ t}$	*application*
$\mathtt{v}$ ::=		*values:*
	$\lambda\mathtt{x.t}$	*abstraction value*

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

$$\frac{\mathtt{t}_1 \longrightarrow \mathtt{t}_1'}{\mathtt{t}_1\ \mathtt{t}_2 \longrightarrow \mathtt{t}_1'\ \mathtt{t}_2} \quad \text{(E-APP1)}$$

$$\frac{\mathtt{t}_2 \longrightarrow \mathtt{t}_2'}{\mathtt{v}_1\ \mathtt{t}_2 \longrightarrow \mathtt{v}_1\ \mathtt{t}_2'} \quad \text{(E-APP2)}$$

$$(\lambda\mathtt{x.t}_{12})\ \mathtt{v}_2 \longrightarrow [\mathtt{x} \mapsto \mathtt{v}_2]\mathtt{t}_{12} \quad \text{(E-APPABS)}$$

Figure 5-3: Untyped lambda-calculus (λ)

Church booleans

```
tru  := λt. λf. t
fls  := λt. λf. f
test := λl. λm. λn. l m n
```

Church booleans

```
tru  := λt. λf. t
fls  := λt. λf. f
test := λl. λm. λn. l m n
```

	`test tru v w`	
$=$	`(λl. λm. λn. l m n) tru v w`	by definition
$\longrightarrow$	`(λm. λn. tru m n) v w`	reducing the underlined redex
$\longrightarrow$	`(λn. tru v n) w`	reducing the underlined redex
$\longrightarrow$	`tru v w`	reducing the underlined redex
$=$	`(λt.λf.t) v w`	by definition
$\longrightarrow$	`(λf. v) w`	reducing the underlined redex
$\longrightarrow$	`v`	reducing the underlined redex

Church Booleans: exercises

```
tru  := λt. λf. t
fls  := λt. λf. f
test := λl. λm. λn. l m n
```

Exercise 1.1. Implement logical **or** and **and** functions.

Church numerals: increment

```
c0 := λs. λz. z
c1 := λs. λz. s z
c2 := λs. λz. s (s z)
c3 := λs. λz. s (s (s z))

inc := λn. λs. λz. s (n s z)
```

Exercise 1.2. Find another way to implement **`inc`**.

Church numerals: addition and multiplication

```
c0 := λs. λz. z
c1 := λs. λz. s z
c2 := λs. λz. s (s z)
c3 := λs. λz. s (s (s z))

plus  := λm. λn. λs. λz. m s (n s z)
times := λm. λn. m (plus n) c0
```

Exercise 1.3. Implement **`times`** without using **`plus`**.

Exercise 1.4. Define a term for raising one number to the power of another.

Exercise 1.5. Define a term that checks whether a given term is zero.

Church numerals: subtraction

```
zz = pair c0 c0
ss = λp. pair (snd p) (plus c1 (snd p))
prd = λm. fst (m ss zz)
```

Exercise 1.6. Use **prd** to define subtraction.

Exercise 1.7. Approximate time complexity of **prd**.

Exercise 1.8. Write function **equal** that tests whether two numbers are equal.

Exercise 1.8*. See [TaPL, Exercise 5.2.8].

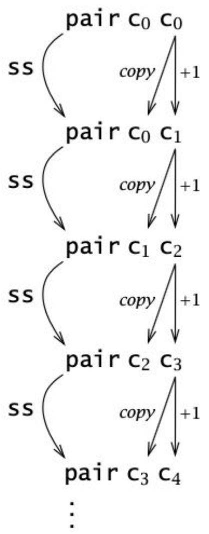


Figure 5-1: The predecessor function's "inner loop"

Enriching the calculus

$\mathbb{B}$ $\mathbb{N}$ *(untyped)* — *Extends* **B** *(3-1)*

New syntactic forms

t	::=	...	*terms:*
		0	*constant zero*
		succ t	*successor*
		pred t	*predecessor*
		iszero t	*zero test*
v	::=	...	*values:*
		nv	*numeric value*
nv	::=		*numeric values:*
		0	*zero value*
		succ nv	*successor value*

New evaluation rules $\boxed{\mathtt{t} \longrightarrow \mathtt{t}'}$

$$\frac{\mathtt{t}_1 \longrightarrow \mathtt{t}_1'}{\mathtt{succ}\ \mathtt{t}_1 \longrightarrow \mathtt{succ}\ \mathtt{t}_1'} \quad \text{(E-Succ)}$$

$$\mathtt{pred}\ 0 \longrightarrow 0 \quad \text{(E-PredZero)}$$

$$\mathtt{pred}\ (\mathtt{succ}\ \mathtt{nv}_1) \longrightarrow \mathtt{nv}_1 \quad \text{(E-PredSucc)}$$

$$\frac{\mathtt{t}_1 \longrightarrow \mathtt{t}_1'}{\mathtt{pred}\ \mathtt{t}_1 \longrightarrow \mathtt{pred}\ \mathtt{t}_1'} \quad \text{(E-Pred)}$$

$$\mathtt{iszero}\ 0 \longrightarrow \mathtt{true} \quad \text{(E-IszeroZero)}$$

$$\mathtt{iszero}\ (\mathtt{succ}\ \mathtt{nv}_1) \longrightarrow \mathtt{false} \quad \text{(E-IszeroSucc)}$$

$$\frac{\mathtt{t}_1 \longrightarrow \mathtt{t}_1'}{\mathtt{iszero}\ \mathtt{t}_1 \longrightarrow \mathtt{iszero}\ \mathtt{t}_1'} \quad \text{(E-IsZero)}$$

Figure 3-2: Arithmetic expressions (NB)

Enriching the calculus

$\mathbb{B}$ *(untyped)*

Syntax		
$\mathtt{t}$ ::=		*terms:*
	$\mathtt{true}$	*constant true*
	$\mathtt{false}$	*constant false*
	$\mathtt{if\ t\ then\ t\ else\ t}$	*conditional*
$\mathtt{v}$ ::=		*values:*
	$\mathtt{true}$	*true value*
	$\mathtt{false}$	*false value*

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

$$\mathtt{if\ true\ then\ t_2\ else\ t_3} \longrightarrow \mathtt{t_2} \qquad \text{(E-IfTrue)}$$

$$\mathtt{if\ false\ then\ t_2\ else\ t_3} \longrightarrow \mathtt{t_3} \qquad \text{(E-IfFalse)}$$

$$\frac{\mathtt{t_1} \longrightarrow \mathtt{t_1'}}{\mathtt{if\ t_1\ then\ t_2\ else\ t_3} \longrightarrow \mathtt{if\ t_1'\ then\ t_2\ else\ t_3}} \qquad \text{(E-If)}$$

Figure 3-1: Booleans (B)

Recursion

```
fix := λf. (λx. f (λy. x x y)) (λx. f (λy. x x y))
g := λ fct. λn. if realeq n c0
                  then c1
                  else (times n (fct (prd n)))
factorial := fix g
```

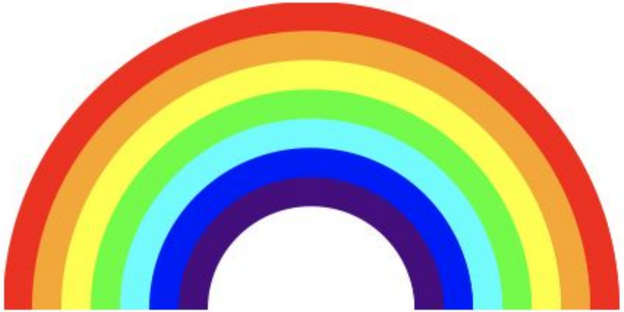
Exercise 1.9. Write down evaluation of **`factorial c3`**.

Exercise 1.10. Why did we use a primitive **`if`** in the definition of **`g`**, instead of the Church-boolean **`test`** function on Church booleans? Show how to define the **`factorial`** function in terms of **`test`** rather than **`if`**.

Exercise 1.11*. Use **`fix`** and the encoding of lists from Exercise 1.8 to write a function that sums lists of Church numerals.

Homework

1. Install **DrRacket** https://download.racket-lang.org
2. Read **Quick: An Introduction to Racket with Pictures** https://docs.racket-lang.org/quick/index.html
3. Read about **Racket Essentials 2.1–2.2** https://docs.racket-lang.org/guide/to-scheme.html
4. Test yourself by implementing a program that renders a rainbow:



References

1. TaPL 5.2