Informatics 1A

Computation and Logic 1

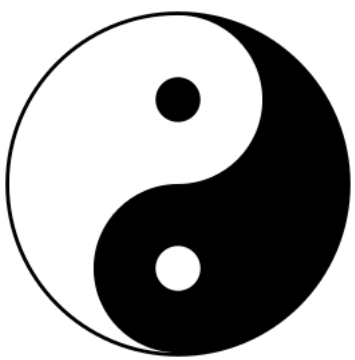
INF1A-CL

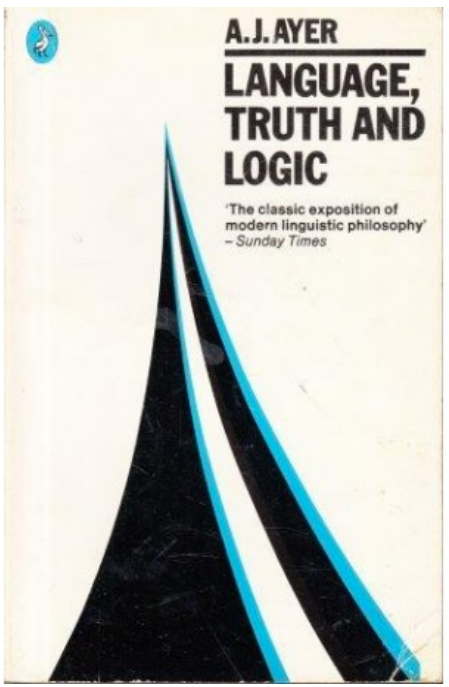
the type Bool

truth
logic
language

Michael P. Fourman

@mp4man





Operations `data Bool = False | True`

```
(&&) :: Bool -> Bool -> Bool    | infixr 3
```

Boolean "and"

```
(||) :: Bool -> Bool -> Bool    | infixr 2
```

Boolean "or"

```
not :: Bool -> Bool
```

Boolean "not"

INF1A-

the type Bool

`True`	$\top$
`False`	$\bot$
`not`	$\neg$
`&&`	$\wedge$
`||`	$\vee$

some Boolean Operations

```
not :: Bool -> Bool -- not ¬
not False = True
not True = False
```

p	¬
⊥	⊤
⊤	⊥

```
(&&) :: Bool -> Bool -> Bool -- and ∧
True && True = True
_    && _    = False
```

∧	⊥	⊤
⊥	⊥	⊥
⊤	⊥	⊤

```
(||) :: Bool -> Bool -> Bool -- or ∨
False || False = False
_     || _     = True
```

∨	⊥	⊤
⊥	⊥	⊤
⊤	⊤	⊤

```
or  :: [Bool] -> Bool -- OR  ⋁
and :: [Bool] -> Bool -- AND ⋀
```

INF1A-

the type Bool

```
or  [] = False
and [] = True
```

Operations

```
(&&) :: Bool -> Bool -> Bool    infixr 3
```

Boolean "and"

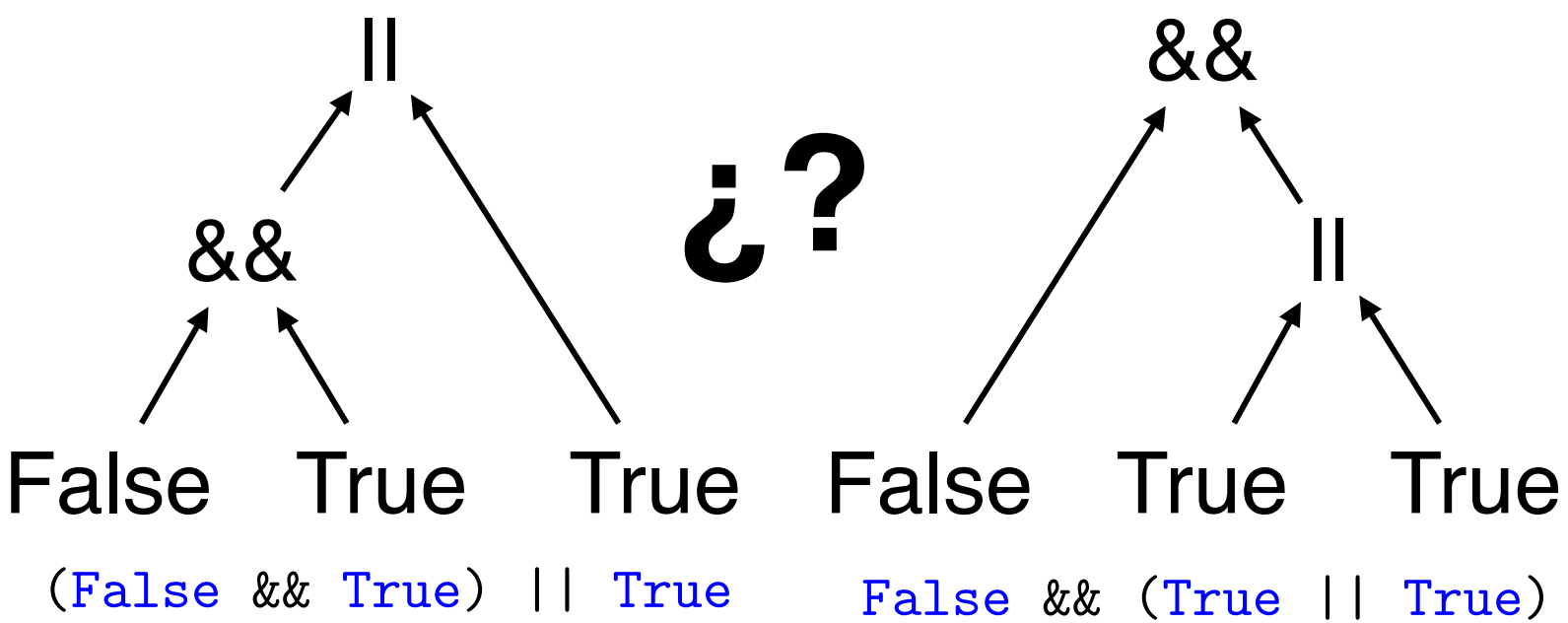
```
(||) :: Bool -> Bool -> Bool    infixr 2
```

Boolean "or"

```
not :: Bool -> Bool
```

Boolean "not"

```
False && True || True
```



An *expression* is a **tree** that describes a computation

Operations

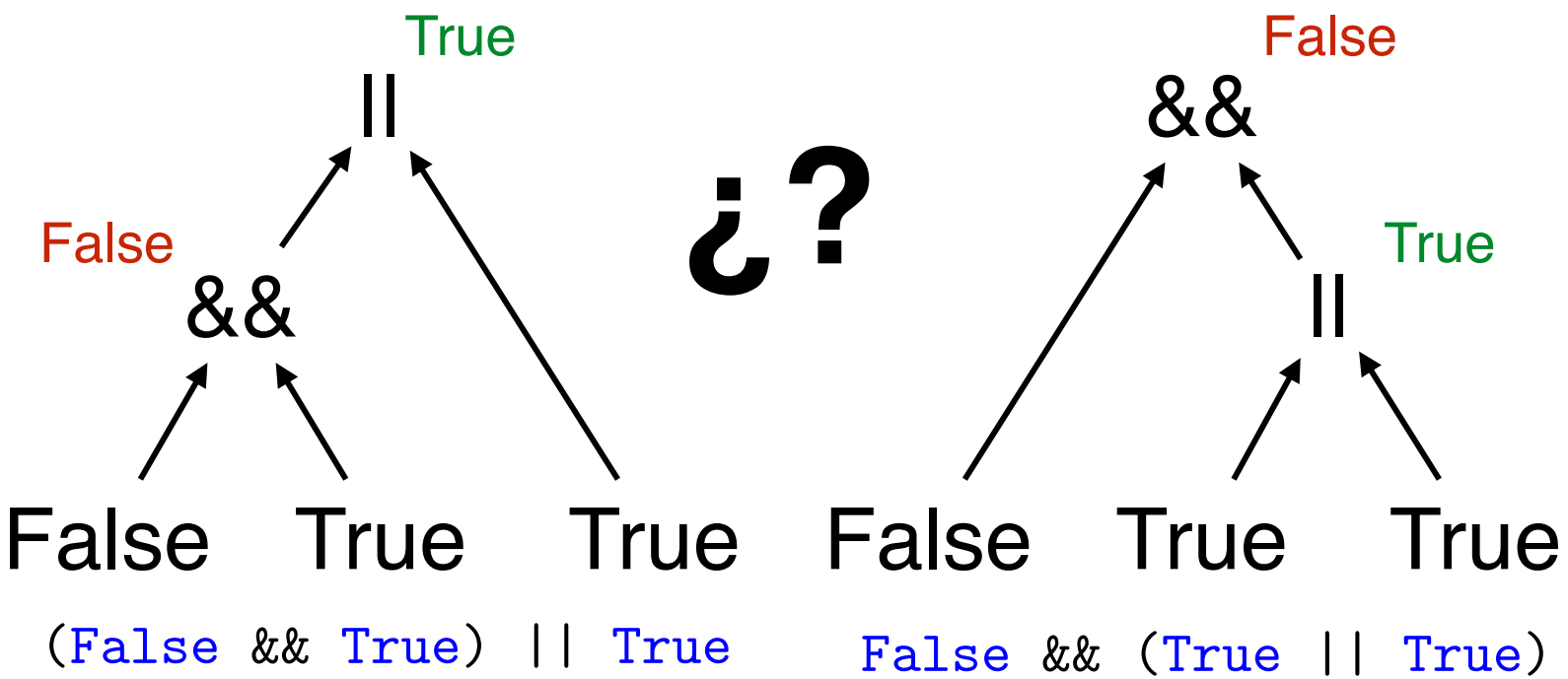
```
(&&) :: Bool -> Bool -> Bool    infixr 3
```

Boolean "and"

```
(||) :: Bool -> Bool -> Bool    infixr 2
```

Boolean "or"

```
False && True || True
```



Operations

```
(&&) :: Bool -> Bool -> Bool    | infixr 3
```

Boolean "and"

```
(||) :: Bool -> Bool -> Bool    | infixr 2
```

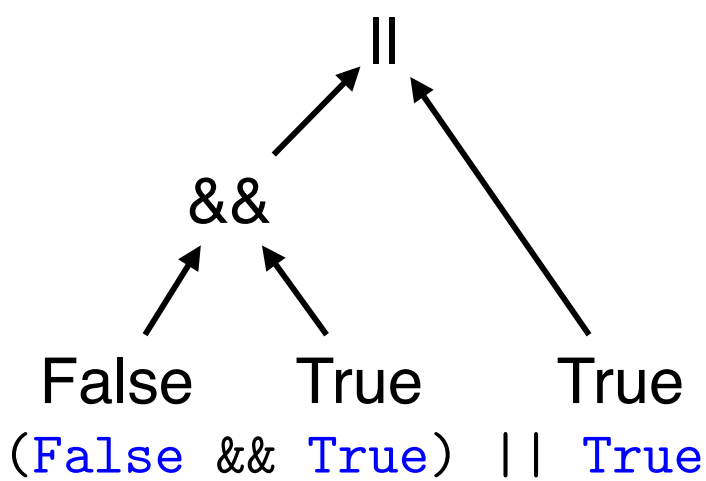
Boolean "or"

```
Prelude> False && True || True
True
Prelude> False && (True || True)
False
Prelude> (False && True) || True
True
```

a && b || c

An expression is a **tree** that describes a computation

(a && b) || c



alternative implementations

```
not :: Bool -> Bool
not a = if a then False else True

(&&) :: Bool -> Bool -> Bool
a && b = if a then b else True

(||) :: Bool -> Bool -> Bool
a || b = if a then True else b

or  :: [Bool] -> Bool
or (x : xs) = if x then True else or xs
or []       = False

and :: [Bool] -> Bool
and (x : xs) = if x then and xs else False
and []       = True

ite :: Bool -> Bool -> Bool -> Bool
ite a b c = if a then b else c
```