Universitat de Girona
Escola Politècnica Superior

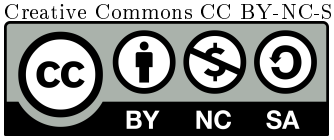
Paradigmes i llenguatges de programació

Curs 2024/25

GEINF

PAC4

Juan José Gómez Villegas, u1987338@campus.udg.edu
Guillem Pozo Sebastián, u1972840@campus.udg.edu



(24 de maig de 2025)

Índex

1 λ-càlcul clàssic

1.1 Tipus

El tipus de dades **LT** ens permet representar λ-termes, i la definició és exactament la gramàtica que diu el que és un λ-terme.

```
data LT = Va String | Ap LT LT | Ab String LT

-- definim la forma de mostrar un lambda terme
instance Show LT where
    show (Va a) = a
    show (Ap t1 t2) = "(" ++ show t1 ++ "␣" ++ show t2 ++ ")"
    show (Ab a t1) = "(\\" ++ a ++ ".␣" ++ show t1 ++ ")"

-- definim la idea d'equivalencia de dos lambda termes
instance Eq LT where
    (==) (Va a) (Va b) = a == b
    (==) (Ap t1 t2) (Ap t1' t2') = (&&) (t1 == t1') (t2 == t2')
    (==) (Ab _ t1) (Ab _ t2) = t1 == t2
    (==) _ _ = False
```

En el codi també es veu com hem fet per tal que el **tipus LT** sigui una instància de les classes **Show** i **Eq**, definint així com es mostraran els λ-termes, i que vol dir que dos λ-termes siguin equivalents.

També hem definit el tipus de dades **Substitució**, que ens ha permet implementar la funció substitució tal i com diu la teòria, és a dir, com un operador que ens permet reemplaçar ocurrències de variables per termes, evitant captura de variables lliures. Per tant, el tipus **Substitució** serà una variable v per un terme M', i la funció **subst** una **Substitució** sobre un terme M, tot junt representarà: $M[v->M']$.

```
data Substitucio = Sub String LT
```

1.2 Funcions auxiliars

1.3 Funcions principals

```
-- freeAndboundVars, donat un LT retorna una tupla amb una llista de freeVars i una ll
freeAndboundVars :: LT -> ([String],[String])
freeAndboundVars t = freeAndboundVarsAux t [] []

-- freeAndboundVarsAux, funcio que construeix una tupla amb dues llistes que contingui
freeAndboundVarsAux :: LT -> [String] -> [String] -> ([String],[String])
freeAndboundVarsAux (Va a) freeVars boundVars = if a `elem` boundVars then (freeVars,b
freeAndboundVarsAux (Ab a t1) freeVars boundVars = (freeAndboundVarsAux t1 freeVars (a
freeAndboundVarsAux (Ap t1 t2) freeVars boundVars = (freeAndboundVarsAux t1 freeVars b

-- subst, donat un LT i una Substitucio, retorna el mateix LT al que se li ha aplicat
subst :: LT -> Substitucio -> LT
subst t s = substAuxInt t s (freeAndboundVars t)

--substAuxInt, funcio intermedia on comprovarem que no es produira cap captura de cap
substAuxInt :: LT -> Substitucio -> ([String],[String]) -> LT
substAuxInt t (Sub v t') l = if (ltPertanyA t' (fst l)) || (ltPertanyA t' (snd l)) the

-- substAux, el mateix subst pero rebent tambe la tupla amb les llistes de variables l
substAux :: LT -> Substitucio -> ([String],[String]) -> LT
substAux (Va a) (Sub v t') l = if a == v && a `elem` (snd l) then t' else (Va a)
substAux (Ab a t1) (Sub v t') l = if a == v then (Ab (getVar t') (substAux t1 (Sub v t
substAux (Ap t1 t2) (Sub v t') l = (Ap (substAux t1 (Sub v t') l) (substAux t2 (Sub v

-- esta_normal, diu si LT ja esta en forma normal
esta_normal :: LT -> Bool
```

```
esta_normal (Va a) = True
esta_normal (Ap (Ab _ _) _) = False
esta_normal (Ab _ t1) = (esta_normal t1)
esta_normal (Ap t1 t2) = (&&) (esta_normal t1) (esta_normal t2)

-- beta_redueix, rep un LT que sigui un redex, i el resol
beta_redueix :: LT -> LT
beta_redueix (Ap (Ab v t1) t2) = substAuxInt t1 (Sub v t2) (freeAndboundVars (Ab v t1)

-- redueix_un_n, rep un LT, i retorna el LT resultant d'aplicar la primera beta-reducc
redueix_un_n :: LT -> LT

-- redueix_un_a, rep un LT, i retorna el LT resultant d'aplicar la primera beta-reducc
redueix_un_a :: LT -> LT

-- l_normalitza_n, rep un LT, i retorna una llista de LT's que sigui una sequencia de
l_normalitza_n :: LT -> [LT]

-- l_normalitza_a, rep un LT, i retorna una llista de LT's que sigui una sequencia de
l_normalitza_a :: LT -> [LT]

-- normalitza_n, rep un LT, i retorna una tupla amb el nombre de passos, mes el LT en
normalitza_n :: LT -> (Int,LT)

-- normalitza_a, rep un LT, i retorna una tupla amb el nombre de passos, mes el LT en
normalitza_a :: LT -> (Int,LT)
```

2 Extra: Notació *de Bruijn*

Per **la notació de Bruijn** hem definit un altre tipus de dades, que tambe es instancia de les classes **Show** i **Eq**.

```
data LTdB = VadB Int | ApdB LTdB LTdB | AbdB LTdB

-- que tambe sera instancia de la classe Show i Eq
instance Show LTdB where
    show (VadB a) = show a
    show (ApdB t1 t2) = "(" ++ show t1 ++ "␣" ++ show t2 ++ ")"
    show (AbdB t1) = "(\\" ++ ".␣" ++ show t1 ++ ")"

-- definim la mateixa idea d'equivalencia que teniem en els lambda termes pels lambda
instance Eq LTdB where
    (==) (VadB a) (VadB b) = a == b
    (==) (ApdB t1 t2) (ApdB t1' t2') = (&&) (t1 == t1') (t2 == t2')
    (==) (AbdB t1) (AbdB t2) = t1 == t2
    (==) _ _ = False
```