TD n°2 – Compte Rendu

1. Traiter le cas d’un terme

*;; derivation du terme a par rapport à la variable x*

(defun deriv-term (a x) (**if** a (**if** (equal a x) 1 0)))

1. Traiter une expression linéaire composée d’un seul type d’opérateur, par exemple l’addition

(defun deriv-mul (e x)

(**if** (atom e)

(deriv-term e x)

(list '+

(list '\* (deriv-mult (cadr e) x) (caddr e))

(list '\* (deriv-mult (caddr e) x) (cadr e)))))

1. Traiter une expression quelconque

(defun deriv (exp)

(cond

((atom exp) (**if** (equal exp var) 1 0))

((equal (car exp) '+) (deriv-sum exp var))

((equal (car exp) '-) (deriv-sub exp var))

((equal (car exp) '\*) (deriv-mul exp var))

((equal (car exp) '/) (deriv-div exp var))

(t (error "exp non traitée"))))

Bien évidemment, les fonctions deriv-\* ont été modifies, de manière a ce qu’elles repassent par derive, par exemple (pour deriv-sum)

(defun deriv-sum (e x)

(list (car e) (**deriv** (cadr e) x) (**deriv** (caddr e) x))))

1. Fonction de simplification

(defun simplify (exp)

(**if** (atom exp)

exp

(**if** (and (atom (cadr exp)) (atom (caddr exp)))

(cond

((equal (car exp) '+) (cond

((equal (cadr exp) '0) (caddr exp))

((equal (caddr exp) '0) (cadr exp))

(T (list (cadr exp) (car exp) (caddr exp)))))

((equal (car exp) '\*) (cond

((OR (equal (cadr exp) '0) (equal (caddr exp) '0)) '0)

((equal (cadr exp) '1) (caddr exp))

((equal (caddr exp) '1) (cadr exp))

(T (list(cadr exp) (car exp) (caddr exp))))))

(list (simpl (cadr exp)) (car exp) (simpl (caddr exp))))))