מחשבון נגזרות (חלקי)

אייל קובי
רון שחר

הערה לגבי המימוש :

במימוש של הקודים שלנו כתבנו מקרה בסיס ל-"x" אך לא כתבנו מקרים למקרי הבסיס האחרים מהקבוצה:

{"(x^n)" | n>=2}

(הכוונה ל-n טבעי).

נסמן ב-A את קבוצת הפונקציות האלמנטריות שהגדרתם באינדוקציה, ונסמן ב-B את קבוצת הפונקציות האלמנטריות שמוצגות אצלינו באינדוקציה. נראה ש-A=B:

B מוכל ב-A:

טריוואילי – הורדנו חלק מהבסיס ולכן בהכרח לכל פונקציה שלה קיימת סדרת יצירה ב-B קיימת גם סדרת יצירה ב-A.

A מוכל ב-B:

בבסיס של הקבוצה המוגדרת באינדוקציה A קיימים כל המחרוזות מהצורה:

{"(x^n)" | n>=2}

דבר זה לא קיים בבסיס של B, נראה שכל איבר מהקבוצה ניתן ליצירה ב-B ולכן (באופן טרנזיטיבי אם תרצו) כל מה שניתן ליצירה ב-A. נראה סדרת יצירה ב-B:

phi1. x { x in B }
phi2. n { x in R and B contains R }
phi3. (x^n) { f5(phi1,phi2) }

מהו קלט תקין?

- **כל מחרוזת שניתן ליצור לה סדרת יצירה מהבסיס שהגדרנו בשקופית הקודמת**

- **נגדיר באינדוקציה קלטים תקינים T :**

B = ℝ ∪ { x, sin(x), cos(x), tg(x), arcsin(x), arccos(x), arctg(x), exp(x), ln(x) }

F = { f0, f1, f2, f3, f4, f5 }

∀x ∈ T : f0(x) = (-x) , ∀x,y ∈ T : f1(x,y) = (x+y) ,∀x,y ∈ T : f2(x,y) = (x-y) ,

∀x,y ∈ T : f3(x,y) = (x*y) ,∀x,y ∈ T : f4(x,y) = (x/y) ,∀x,y ∈ T : f5(x,y) = (x^y)

- **למשל הקלטים הבאים :**

$\frac{x+x}{x}$ → *((x+x)/x)*

$\cos(x)$ → *cos(x)*

$-\frac{x^{\ln(x)} \cdot \sin(x)}{\arccos(x)+5x}$ → *(-(((x^ln(x))*sin(x))/(arccos(x)+(5*x))))*

איך עובד האלגוריתם?

איך עובדת קריאת המחרוזת (נכון גם לDiff וגם לEval)

ראשית הקוד בודק האם המחרוזת שייכת לאחד ממקרי הבסיס ובמידה וכן מחזירה פלט מתאים.
אחרת מתבצעת בדיקה האם התו הראשון הוא ")" והתו האחרון הוא "("
במידה והתשובה היא לא תיזרק שגיאה סינטקטית.

במידה והתשובה היא כן והמחרוזת מתחילה בתוים "-)" כלומר יש מינוס בהתחלה, הפונקציה תקרא לעצמה ללא שני התווים הראשונים (המינוס והסוגרים) וללא התו האחרון (גם סוגרים).
לאחר מכן היא תשלח את הפונקציה לרקורסיה בלי המינוס. והפלט שחזר מהרקורסיה יטופל בהתאם למינוס.

במצב שהמחרוזת לא מתחילה במינוס (או נשלחת בלי המינוס) הפונקציה תעבור על המחרוזת (בלי הסוגרים שלה) ותוסיף 1+ למונה על כל סוגר שמאלי ")" ותוריד 1- למונה על כל סוגר ימני "(" ברגע שהמאזן של המונה הוא אפס והתו שאותו אנו קוראים הוא מהסימנים +, -, *, / , ^ אז נחשב את הערך של המחרוזת לפני הסימן ואחרי הסימן בהתאם לסימן.

איך האלגוריתם עובד - Diff

במקרה הבסיס תחושב הנגזרת ישירות, לדוגמא Diff(sin(x)) = cos(x) .

במקרה של צעד "-" (מינוס חד מקומי) נחשב כך : Diff(-f(x)) = -Diff(f(x)) .

במקרה של צעד "+" (פלוס) נחשב כך : Diff(f(x)+g(x)) = Diff(f(x)) + Diff(g(x)) .

במקרה של צעד "-" (מינוס דו מקומי) נחשב כך : Diff(f(x)-g(x)) = Diff(f(x)) - Diff(g(x)) .

במקרה של צעד "*" (כפל) נחשב כך : Diff(f(x)*g(x)) = Diff(f(x)) * g(x) + f(x) * Diff(g(x)) .

במקרה של צעד "/" (חילוק) נחשב כך : Diff(f(x)/g(x)) = (Diff(f(x)) * g(x) - f(x) * Diff(g(x))) / (g(x)^2) .

במקרה של צעד "^" (חזקה) נחשב כך : Diff(f(x)^g(x)) = (f(x) ^ g(x)) * (Diff(g(x)) * ln(f(x)) + (g(x) * Diff(f(x)) / f(x)) .

איך האלגוריתם עובד - EVAL

במקרה הבסיס תחושב הערך ישירות, לדוגמא Eval(sin(x)) = sin(x) .
או Eval(sin(x),0) = 0

במקרה של צעד "-" (מינוס חד מקומי) נחשב כך : Eval(-f(x)) = -Eval(f(x)) .

במקרה של צעד "+" (פלוס) נחשב כך : Eval(f(x)+g(x)) = Eval(f(x)) + Eval(g(x)) .

במקרה של צעד "-" (מינוס דו מקומי) נחשב כך : Eval(f(x)-g(x)) = Eval(f(x)) - Eval(g(x)) .

במקרה של צעד "*" (כפל) נחשב כך : Eval(f(x)*g(x)) = Eval(f(x)) * Eval(g(x)) .

במקרה של צעד "/" (חילוק) נחשב כך : Eval(f(x)/g(x)) =Eval(f(x)) / Eval(g(x)) .

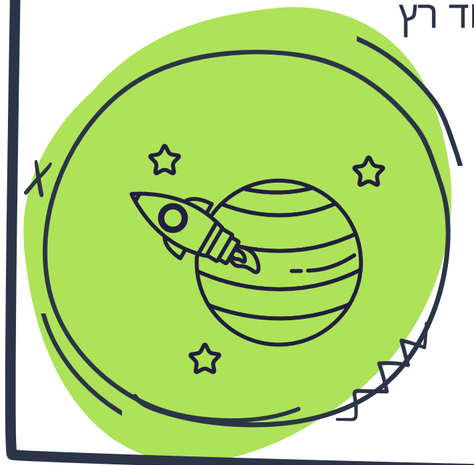
במקרה של צעד "^" (חזקה) נחשב כך : Eval(f(x)^g(x)) = Eval(f(x)) ^ Eval(g(x)) .

יתרונות וחסרונות של כל אחת מהשפות בקוד שלנו

Haskell	Python
הודות ל-haskell ניתן היה לבנות פעם אחת את פונקציית eval כך שסוגה יהיה: [Char] -> Double -> Double. באמצעות סוג זה, במידה והחסרנו את x, השפה באופן אוטומטי יצרה פונקציה f חד-מקומית שמבצעת את תפקידה המבוקש הודות לאופן פעולת השפה יתרון על שפת פייתון.	נוצר צורך להפרדה ידנית בין המקרה בו יש קלט x לפונקציית eval לבין מקרה בו אין קלט כזה (ויצירת ביטוי למבדה בהתאם) זה חסרון לעומת הסקל.
על-מנת לפרק את המחרוזת לפי קשר מוביל היה צורך ליצור פונקציה באינדוקציית מבנה עם ארבעה קלטים ו-tuple בעל שלושה ארגומנטים. זה קשה יותר למימוש ולקריאה (עשינו זאת לפני שלמדנו מונדות וכתיבה אימפרטיבית בשפת Haskell). חיסרון לעומת פייתון	את פירוק המחרוזת לפי קשר מוביל ניתן היה לבצע בקלות בלולאת for פשוטה ושימוש ב slice יתרון לעומת הסקל
בשפת Haskell נאלצנו לעבור תנאי תנאי ולא הייתה לנו דרך לקצר ולעשות זאת יפה יותר (לפחות ממה שאנחנו הכרנו) חסרון לעומת פייתון	בשפת פייתון השתמשנו ב Map דבר ששימש עברנו כמעין "switch case" וחסך לנו הרבה מקרים של if else קוד נקי ומסודר יותר יתרון לעומת הסקל
בשפת Haskell כאשר רצינו לבדוק האם איבר מסויים שייך לקבוצה מסויימת (לדוגמא לקבוצה {+,-,*,/,^} - קבוצת תווי היצירה) היינו צריכים לעשות זאת הפונקציה בוליאנית (עם הקשר הלוגי or כמובן..) חסרון לעומת פייתון	בשפה פייתון כאשר רצינו לבדוק שייכות לקבוצה (לדוגמא לקבוצה {+,-,*,/,^} - קבוצת תווי היצירה) ניתן היה להשתמש ב-set שבפייתון, ובאופרטור in, כתיבה שהייתה הרבה יותר user friendly לנו כמהנדסי תוכנה שלמדו תורת הקבוצות. יתרון לעומת הסקל

אופטימיזציה

1. רקורסיית זנב - ב-Haskell הקפדנו כמעט בכל פונקציה שרשמנו להשתמש ברקוסיית זנב ולא ברקוסיה פנימית על-מנת לשפר זמן ריצה של הקוד.
2. שימוש מופחת ב-else - כאשר בסוף ה-branch של if הייתה פעולת return דאגנו להימנע מ-else כיוון שפעולת ה-return מונעת מאיתנו להגיע ל-branch של ה-else גם ככה.
3. פונקציית העזר numbersHandeller - פונקציה זו דאגה ישירות לחשב מחרוזות שמייצגות מספר בפעולות של +,-,*,/,^. דבר זה סייע לנו להתמודד עם מחרוזות קצרות יותר וכך נוצר ייעול של ריצת הקוד.
4. פונקציית העזר adishHandeller - פונקציה זו דאגה לטפל במקרים של סקלרים אדישים לפעולת חשבון מסויימת (לדוגמא 1 אדיש כפלי ו-0 אדיש חיבורי). ע"י טיפול באדישים אלו המחרוזות התקצרו במקרי קיצון אלו וכך הקוד רץ במהירות גדולה יותר במקרים אלו.



```
running time on Python:  0.02497863769531 25
```

0.02 Sec

זמן ריצה בפייתון כאשר הקלט הוא באורך 1000 תווים

```
2.0389977s
```

2.04SEC

זמן ריצה בהסקל כאשר הקלט הוא באורך 1000 תווים

Thank you!