digital whisper
Technologic Papers

מספרי צ׳רץ׳

מאת סופי ציאדה

הקדמה

בואו נדבר על מספרים. אבל לא על המספרים ה״רגילים״ שכולנו מכירים, אלא על יצוג אחר של מספרים, מספרים בקידוד צ׳רץ׳. קידוד צ׳רץ׳ היא דרך להציג את המספרים הטבעיים בתור פונקציות מסדר גבוה.

פונקציה מסדר גבוה היא פונקציה שמקבלת פונקציה כלשהי כקלט, או מחזירה פונקציה כלשהי כפלט. או במילים פשוטות יותר - אלה פונקציות שפועלות על פונקציות.

הקידוד מתאים למספר הטבעי n את הפונקציה $C_n(f,x)$ שמוגדרת כך:

C_n היא פונקציה *שמקבלת* שני פרמטרים:

- הראשון, f, הוא פונקציה שמסומנת ב-f וממפה איבר כלשהו מסוג גנרי T, אל איבר כלשהו מאותו הסוג, T.
- הפרמטר השני, x, הוא איבר כלשהו, מאותו הסוג - T.

הפונקציה *מחזירה* את התוצאה של הפעלת f על x, n פעמים בשרשרת - בפעם ראשונה נפעיל את f על x, בפעם השנייה נפעיל את f על התוצאה של השלב הקודם (כלומר $f(x)$), וכן האלה n פעמים.
אם נרצה לכתוב את זה באופן פורמלי נכתוב את זה כך:

$$\begin{aligned} C_n(f,x) &= f(\dots f(x)\dots) \\ &= f \circ f \circ \dots \circ f(x) \\ &= f^n(x) \end{aligned}$$

הופכים את זה לממשי

נסתכל על דוגמה אחת כדי להפוך את זה לפחות מופשט.

נגדיר את f שלנו בתור $f(x) = 2 * x$. מה שאומר שבדוגמאות הבאות, הקבוצה, או הסוג הגנרי, שהזכרנו למעלה, T, יהיה המספרים הממשיים ($\mathbb{R}$).

- הפונקציה C_0 היא הפונקציה שקידוד צ׳רץ׳ מתאים למספר 0. התוצאה של $C_0(f,3)$ תהיה 3. למה? כי הפעלנו את הפונקציה f על 3 אפס פעמים, כלומר, לא הפעלנו, ולכן נשארו עם האיבר המקורי. למעשה, באופן כללי, הפונקציה C_0 תמיד מחזירה את הפרמטר השני שלה כמו שהוא.
- התוצאה של $C_1(f,3)$ היא $f(3)=2*3=6$
- התוצאה של $C_2(f,3)$ היא $f(f(3))=2*(2*3)=12$

אז הבנו איך הקידוד ממפה מספר טבעי n לפונקציה C_n. אבל זה רק כיוון אחד.

לשמחתנו, המיפוי ההפוך אפילו יותר פשוט. אם יש לנו פונקציה C_h, שידוע לנו שהיא מקודדת מספר טבעי בקידוד צ׳רץ׳, ואנחנו רוצים לגלות מהו אותו ה-h, נחשב את הפונקציה עם $f(x)=x+1$ בתור פרמטר ראשון ו-0 בתור פרמטר שני.

למה זה עובד? כי C_h תפעיל את f על 0 בסך-הכל h פעמים, כלומר, תוסיף ל-0 את המספר 1 h פעמים, ולכן התוצאה תהיה h.

אז המיפוי עובד לשני הכיוונים ☺

מטא-מטא פונקציות

בואו נסמן את הפונקציה שמתאימה למספר n את הפונקציה C_n בתור C_{enc} (encode, לא encrypt ☺). את הפונקציה שעושה את המיפוי ההפוך, או הפענוח, נסמן ב-C_{dec}.

אמרנו שהפונקציות C_n, או במילים אחרות מספרי צ׳רץ׳, מייצגות מספרים, אז בואו נגדיר פעולות חיבור וכפל על המספרים האלה. המטרה היא להגדיר פעולות בין המספרים האלה שישמרו על המיפוי נכון. כלומר, נרצה שיתקיים:

$$C_{enc}(n+m)=C_{enc}(n)+C_{enc}(m)$$

הסימון פה טיפה טריקי. מה שקורה מצד שמאל זה חיבור רגיל בין שני טבעיים n, ו-m. על תוצאת החיבור מפעילים את C_{enc} ומקבלים את הפונקציה C_{n+m}.

מה שקורה מצד ימין זה שמפעילים את C_{enc} על כל מספר טבעי בנפרד, מקבלים שתי פונקציות C_n, ו-C_m ומבצעים בינהן חיבור. זה לא חיבור בין מספרים, אלא בין פונקציות. ואת אופן החיבור הזה, אנחנו צריכים להגדיר.

למעשה, התוצאה של החיבור C_n+C_m צריכה להיות הפונקציה C_{n+m}. כלומר, היא מקבלת כפרמטרים פונקציה f וערך התחלתי x ומפעילה את f על x סה״כ $n+m$ פעמים. באותה הצורה נגדיר כפל. התוצאה של C_n*C_m היא הפונקציה C_{n*m}.

מה שאלונזו צ׳רץ׳ רצה להראות בקידוד הזה, זה שאפשר לפתור כל בעיה חישובית דרך שימוש בפונקציות בתור טיפוס נתונים בסיסי, בלי לפנות בכלל למספרים. כמובן שלא משתמשים באופן היצוג הזה בפרקטיקה. הרבה יותר פשוט וזול לייצג בזכרון מספר ביצוג בינארי מאשר בתור פונקציה.

אבללל, זה לא ימנע מאיתנו לעשות את זה בכל זאת, כי זה יעזור לנו להבין איך בפועל עובדים חיבור וכפל בקידוד הזה!

קצת הסקלית לנשמה

אנחנו הולכים להגדיר את הקידוד שלנו ב-Haskell, שזו שפה נהדרת למטרה הזאת, כי פונקציות הן first-class citizens אצלה, וקל מאוד לכתוב פונקציות שמטפלות בפונקציות. אני יודעת שהיא בקושי בשימוש, אז אני מבטיחה להסביר כל מילה בקוד.

נגדיר טיפוס נתונים חדש בשם CNumber.
למעשה, CNumber הוא לא טיפוס חדש, אלא מעטפת לז׳אנר של טיפוסים:
הוא מתאים לכל פונקציה ש:

- מקבלת כפרמטרים:
  - פונקציה שממפה איבר מ-T לאיבר ב-T , ה-f שאנחנו מכירים ממקודם
  - איבר ב-T , או x
- ומחזירה איבר ב-T.

ה-T הזה יכול להיות כל טיפוס נתונים, string, float32 וכו׳.
כלומר T , הוא פרמטר גנרי בפונקציה ש-CNumber עוטף.

נכתוב את זה בהסקלית:

```
newtype CNumber = Nr (forall t. (t -> t) -> t -> t)
```

אנחנו מגדירים פה את CNumber ואומרים שהבנאי שלו, שנקרא Nr, מקבל פונקציה גנרית (שהמשתנה הגנרי שלה הוא t) מהצורה שהגדרנו מקודם.

בהסקל, הטיפוס `a -> b -> c` מתאר פונקציה שמקבלת כפרמטר ראשון איבר מסוג a וכפרמטר שני איבר מסוג b ומחזירה איבר מסוג c. אז מה שהגדרנו פה היא פונקציה ש:

- הפרמטר הראשון שלה הוא `(t -> t)`, כלומר פונקציה שלוקחת ומחזירה איבר מסוג t.
- הפרמטר השני שלה הוא ערך מהסוג t.
- והיא מחזירה ערך מהסוג t.

אז מה שהגדרנו כאן זה תבנית למספרי צ׳רץ׳ - C_n. זה לא אומר שכל הפונקציות שמתאימות לתבנית *בהכרח* מתארות מספרים בקידוד צ׳רץ׳ - התבנית תתאים בבאופן כללי לפונקציות שמקבלות פונקציות ואיבר, ומחזירות איבר.

בואו נשתמש בהגדרה בשביל להגדיר מספרים:

```
zero = Nr (\ f x -> x )
```

כמו שאמרנו מקודם C_0 , לא מפעילה את f בכלל ומחזירה את x כמו שהוא.
שזה בדיוק מה שכתוב פה - אנחנו מעבירים ל-Nr פונקציית למדא, שמחזירה את הערך שהיא מקבלת.

```
one = Nr (\ f x -> f x )
```

C_1 מפעילה את f על x פעם אחת.

```
two = Nr (\ f x -> f (f x) )
```

באותו האופן C_2 , מפעילה פעמיים.

אז הצלחנו לייצג את הקידוד בכיוון הראשון, נעבור לכיוון השני - מפונקציה למספר.
נניח שיש לנו עצם מסוג CNumber, וידוע שהוא קידוד של מספר טבעי כלשהו. איך נוכל לשחזר את המספר שהוא מייצג? כבר פתרנו את הבעיה הזאת מקודם, ובהסקל נכתוב את זה ככה:

```
eval :: CNumber -> Int
eval (Nr Cn) = Cn (+1) 0
```

הפונקציה eval, מקבלת פונקציית צ׳רץ׳, מסוג Nr, וקוראת לה עם הפרמטרים (+1), שמקבילה ל-$f(x) = x + 1$ בהסקל, והמספר 0. התוצאה היא המספר הטבעי שהפונקציה מייצגת.

לפני שנממש את הפונקציה שמחברת שני מספרי צ׳רץ׳, נממש אחת פשוטה יותר בשם succ. מה שהיא עושה זה לקבל עצם מסוג CNumber, ולהחזיר את העצם הבא אחריו ביצוג. כלומר, אם היא קיבלה את one/C_1 היא תחזיר את two/C_2. הפונקציה הזו מקבילה לפונקציית העוקב של המספרים הטבעיים.

```
succ :: CNumber -> CNumber
succ (Nr Cn) = Nr (\ f x -> f (Cn f x) )
```

נניח ש-succ מקבלת CNumber שמתאר את הטבעי n.
היא מחזירה כתוצאה פונקצייה חדשה, שהיא גם מספר צ׳רץ׳ – כי היא מקבלת פונקציה f ופרמטר x. הפונקציה החדשה תשתמש ב-Cn, שהיא פונקציית צ׳רץ׳, כדי להפעיל את f על x n פעמים (כי זה בהגדרה מה ש-Cn, או C_n, עושה), ואז תפעיל עוד פעם אחת את f על התוצאה הסופית. וכך בעצם קיבלנו את העצם מסוג CNumber שמתאר את $n + 1$, או C_{n+1}.

חיבור, חיסור וכפל בצ'רצ'ית

כעת, יהיה לנו הרבה יותר קל להגדיר את פונקציית החיבור. אנחנו רוצים פונקציה בשם add שתקבל שני עצמים מסוג CNumber ותחזיר את ה-CNumber שמתאר את החיבור של העצמים, לפי חוקי החיבור שהגדרנו למעלה:

```
add :: CNumber -> CNumber -> CNumber
add (Nr Ca) (Nr Cb) = Nr (\ f x -> Ca f (Cb f x))
```

נניח ש-Ca ו-Cb הם מספרי צ'רץ' שמייצגים את הטבעיים a ו-b בהתאמה. הפונקציה שבסוגריים הכי פנימיים משתמשת קודם ב-Cb בשביל להפעיל את f על x , b פעמים. אחר-כך אנחנו משתמשים ב-Ca בשביל להפעיל על התוצאה של הסוגריים הפנימיים את f, עוד a פעמים. בסה"כ אנחנו מפעילים את f על x, $a+b$ פעמים, שזה בדיוק מה שרצינו.

אבל אנחנו מסוגלים לכתוב את זה יותר יפה. יש לנו כפרמטר את הפונקציה Ca שיודעת להפעיל פונקציות על איבר כלשהו a פעמים. למה לא שנשתמש בה בשביל להפעיל את succ על Cb Nr וככה למצוא את ה-CNumber שמתאר את $a+b$?

אם מפעילים את succ על Cb פעם אחת מקבלים את ה-CNumber שמייצג את $b+1$.

ולכן אם נפעיל את succ על Cb a פעמים באמצעות Ca נקבל את ה-CNumber שמייצג את $a+b$:

```
add :: CNumber -> CNumber -> CNumber
add (Nr Ca) (Nr Cb) = Ca succ (Nr Cb)
```

יש דרך אפילו יותר קצרה לכתוב את זה, והיא:

```
add :: CNumber -> CNumber -> CNumber
add (Nr Ca) = Ca succ
```

אבל אנחנו לא מנסים להגיע לקצרנות, אלא לקריאות ☺

נשאר לנו לממש רק פונקציית הכפל! אציין שזה תרגיל נהדר בעיניי וכדאי לכם לנסות לפתור אותו בעצמכם. כמובן שאתם יכולים פשוט להסתכל על התשובה בהמשך, אבל אז יהרס כל הכיף ☺

אנחנו רוצים להשתמש עוד פעם ב-Ca בשביל לחשב את ה-CNumber החדש. נעשה את זה ככה:

```
mult :: CNumber -> CNumber -> CNumber
mult (Nr Ca) (Nr Cb) = Ca (add (Nr Cb) zero
```

נעבור שלב שלב.

- התוצאה של add (Nr Cb) היא פונקציה שמקבלת CNumber כלשהו ומחברת אליו את Nr Cb (דרך הפעלה של succ b פעמים).
- אנחנו משתמשים ב-Ca בשביל להפעיל על zero, ה-CNumber שמייצג את 0, את התוצאה של השלב הקודם a פעמים.
- בכל הפעלה של add (Nr Cb) אנחנו מפעילים את succ b פעמים. אנחנו עושים את זה a פעמים, ולכן הפעלנו את succ בסה"כ $a * b$ פעמים. בעצם, קיבלנו את ה-CNumber שמייצג את $a * b$.

באותה צורה אפשר להגדיר חזקה, וטטרציה.

אני פשוט אראה את הקוד, ואשאיר את ההבנה כתרגיל לקורא ☺

```
mult :: CNumber -> CNumber -> CNumber
mult (Nr Ca) (Nr Cb) = Ca (add (Nr Cb)) zero
```

סיכום

קידוד צ'רץ' מציע דרך לייצוג מספרים באמצעות פונקציות. הרעיון הזה ניתן להרחבה גם לטיפוסים אחרים, כמו בולאנים, מספרים שליליים ואפילו מספרים ממשיים. בעצם, פונקציות הן אבני בנייה ורסטיליות מספיק בשביל להחליף לגמרי נתונים פרימיטיביים. התרגיל המחשבתי הזה עוזר לנו לחשוב מחדש על מה הם מספרים, במהות שלהם, ועל דרכים אלטרנטיביות לייצג אותם.

על המחברת

סופי ציאדה - Senior Full Stack Developer

המאמר נכתב במקור ב-2020 ו[פורסם](#) בבלוג שלי (גם ב[אנגלית](#)), בהשראת [אתגר קוד](#) באתר CodeWars.

[Sophie Saiada](#) · [GitHub](#) · [LinkedIn](#)