מאוניברסליות לאלגנטיות

יוסי גיל
הפקולטה למדעי המחשב
הטכניון - מכון טכנולוגי לישראל

26 באוקטובר 2021

תוכן העניינים

רשימת הטבלאות

רשימת האיורים

1 שפות תכנות אוניברסליות

אנו אומרים ששפת תכנות היא **אוניברסלית** (universal) או **טיורינג-שלמה** אם (Turing-complete)אם ניתן לממש בתוך שפה זו **כל** שפה אחרת. בניסוח מדויק יותר, שפה $\mathcal{L}$ היא אוניברסלית, אם עבור כל שפה תכנות אחרת $\mathcal{Q}$ ניתן לכתוב בשפה $\mathcal{L}$ תכנית, תכנית הקרויה **תכנית אוניברסלית עבור** $\mathcal{Q}$, הממשת את $\mathcal{Q}$ בתוך $\mathcal{L}$. במילים אחרות ישנה בשפה $\mathcal{L}$ לפחות תכנית אחת $\ell_{\mathcal{Q}}$, $\ell_{\mathcal{Q}} \in \mathcal{L}$, המסוגלת להדר, לבצע, או לממש בכל דרך אחרת **כל** תכנית q בשפת $\mathcal{Q}$: במקום **להריץ** (run, or execute) את q בשפת התכנות $\mathcal{Q}$, ניתן להריץ בשפה $\mathcal{L}$ את התכנית $\ell_{\mathcal{Q}}$ על הקלט q, ושתי ההרצות תהיינה **שקולות**.

המימוש של מיני-ליספ נעשה באמצעות תכניות אוניברסליות כאלו. המימוש הראשון של מיני-ליספ נעשה בשפה דמוית C אשר בה נכתבה תכנית אוניברסלית בעבור למיני-ליספ. ברוח המינימליות של מיני-ליספ, תכנית אוניברסלית זו את נושאת השם הקצר במיוחד m והיא מופעלת באמצעות הוא באמצעות המעטפת של מערכת ההפעלה (e.g., system|). Linux on found as bash named shell the

מיני-ליספ זכתה גם למימוש בתכנית אוניברסלית בשפת JavaScipt. כדי להשתמש בתכנית אוניברסלית יש לגלוש באמצעות הדפדפן לדף HTML המכיל בתוכו את התכנית האוניברסלית הזו. מתברר כי המימוש של כל שפת תכנות נעשה באמצעות תכניות אוניברסליות כאלו: מהדר לשפת Java הכתוב בשפת C, מהדר של שפת פסקל הכתוב בשפת מכונה, אינטרפרטר של שפת פרולוג הכתוב בשפת Java, וכו'.

נציין כי כמעט כל שפות התכנות, ובכללן פסקל, C, וגם מיני-ליספ, הינן אוניברסליות. השפות שאינן אוניברסליות הינן מעטות מאוד, וכוללות בתוכן מספר שפות לעיבוד נתונים ובכללן Datalog ו-SQL, וכן שפות מועטות כגון LOOP אשר תוכננו במפורש שלא להיות אוניברסליות.

כדי להבין מדוע מרבית השפות הינן אוניברסליות, נשים לב לכך שכדי להראות ששפה מסוימת $\mathcal{L}$ היא שפה אוניברסלית, מספיק להראות שיש ב-$\mathcal{L}$ תכנית אוניברסלית $\ell_{\mathcal{U}}$ עבור שפה אחת בלבד $\mathcal{U}$, ובלבד שהשפה $\mathcal{U}$

היא אוניברסלית מצידה. כך למשל, אם ידוע ששפת מיני-ליספ היא אוניברסלית, הרי האינטרפר `m` של מיני-ליספ הכתוב בשפת `C` מוכיח כי אף שפת `C` היא אוניברסלית:

> כאמור, האינטרפטר הוא תכנית ℓ_{mLisp} שהוא אוניברסלית בעבור מיני-ליספ, והכתובה בשפת `C`. כיוון שמיני-ליספ היא אוניברסלית הרי לכל שפת תכנות אחרת $\mathcal{Q}$ קיימת תכנית אוניברסלית מתאימה $\ell_{\mathcal{Q}}$ במיני-ליספ. תכנית אוניברסלית $c_{\mathcal{Q}}$ הכתובה בשפת `C` בעבור השפה $\mathcal{Q}$, מתקבלת באמצעות הפעלה האינטרפטר `l` על $\ell_{\mathcal{Q}}$. על $\ell_{\mathcal{Q}}$.
>
> בפרט, נזכור כי האינטרפטר ממומש כתכנית בשפת `C`. תכנית זו ניתן לתקן בקלות כך שמקום להפעיל האינטרפטר על תכנית כלשהי של מיני-ליספ, הוא יוגבל לפעול על התכנית $\ell_{\mathcal{Q}}$, ועליה בלבד. תכנית מתוקנת זו היא בדיוק התכנית אוניברסלית $C_{\mathcal{Q}}$.

התיאור לעיל של המונח "פונקציה אוניברסלית" איננו הגדרה מתימטית, והוא דורש, למשל, ניסוח מדויק יותר של משמעות פעולת "ההרצה", או, ביאור משמעות הטענה בדבר שקילות של שתי הרצות. הגדרה מתימטית של אלו אינה ענין פשוט: ראשיתה בשנות השלושים של המאה העשרים, עוד טרם פיתוח שפות תכנות כלשהן, בעבודות של מתימטקאים ולוגיקאים מפורסמים[1] כגון דוד הילברט (Hilbert), וילהלם אקרמן Ackerman, סטפן קלין (Kleene) קורט גדל (Go□del) וז'ק הרברנד (Hebrand, ושימוש בלוגיקה מתימטית לשם הגדרה תיאורטית של קבוצה של פונקציות מתימטיות הידועה בשם **קבוצת הפונקציות הרקורסיביות הכלליות**, או בקיצור, **הפונקציות הרקורסיביות**. הגדרה זו, סופה שהתפתחה להגדרות מדויקות של מונחים כגון חישוב, הרצה, ושקילות.

דיוק שכזה חורג מגבולות הדיון כאן אשר די לו בהבנה אינטואיטיבית בסיסית של מושגים אלו: הרצה היא הפעלה של תכנית, אשר יכולה לכלול בתוכה גם הידור, ושתי הרצות הובסיסית שקולות אם הפלט שלהן זהה.

המונח חישוב יכלול עבורנו פעולות כגון בדיקה של תנאים וקריאה לפונקציות. אך לא נכלול במונח חישוב פעולות כגון בדיקת מיקום העכבר, או תצוגה גרפית, וזאת מהשיקול הבא:

> אם השפה $\mathcal{Q}$ תומכת בפעולה)כגון תצוגה גרפית(α,כך ש-α אינה יכולה להיות ממומשת באמצעות מבנים אחרים של $\mathcal{Q}$, והשפה $\mathcal{L}$ אינה תומכת ב-α. הרי אמנם לא ניתן לכתוב ב-$\mathcal{L}$ תכנית אוניברסלית עבור $\mathcal{Q}$, אולם, אם ישנו מימוש עבור השפה $\mathcal{Q}$, ניתן לתקן את $\mathcal{L}$ באמצעות הוספת הפעולה α לשפת התכנות.

2 מכונת טיורינג והסברה של טיורינג וצ'רץ'

קושי מובנה נוסף בתיאור המילולי ולא פורמלי של שפות תכנות אוניברסליות, הוא בדרישה כי קיימת בה תוכנית אוניברסלית בשפה עבור כל שפת תכנות אחרת. וכי מה הכוונה במונח המעורפל "כל שפת תכנות אחרת"?

מסקנתנו כי ניתן לזהות שפה תכנות כאוניברסלית על ידי כתיבה של תכנית אוניברסלית אחת בלבד בה עבור שפה אוניברסלית מסוימת אחרת, מצמצמת מעט את השאלה. הרי די לנו לבנות שפת תכנות אוניברסלית אחת, השפה האוניברסלית ה"ראשונית" והראשונה. רק בבניה זו נידרש להפיג את הערפל מעל המונח "כל שפת תכנות אחרת", ולו פעם אחת בלבד.

שפת התכנות הראשונית הזו אינה שפה, אלא מחשב, מחשב פשוט מאוד, אך מחשב שלא נבנה מעולם. מחשב זה ידוע בשם מכונת טיורינג: באותן שנות השלושים בהן עמלו מתימטיקאים ולוגיקאים על הגדרת קבוצות הפונקציות הרקורסיביות, פעל גם אלן טיורינג)Turing(, מתימטיקאי בעצמו, אך גם אחד ממדעני המחשב הראשונים. טיורינג השתמש בהגדרה זו של קבוצת הפונקציות הרקורסיביות ובהתפתחויות אחרות במתימטיקה,

[1] נאמר שמתימטיקאי הוא **מפורסם** אם קיים משפט או מונח מתימטי הקרוים על שמו: מרחבי הילברט, משפט הרברנד, משפטי אי השלמות של גדל, פונקצית אקרמן, אלגברת קלין, מכונת טיורינג, ארכיטקטורת פון-נוימן ומשפט צ'רץ'-רוסר הופכים על כן את Hilbert, Hebrand, Hebrand, Go□del, Ackerman, Kleene, Turing, von Neumann, Church ו-Rosser למתימטיקאים מפורסמים.

לשם ניסוח מדויק של המונח **חישוב**, ותיאור מדויק כיצד חישוב יכול להעשות הן באופן מוכני בידי אדם והן בידי מכונה.

באותה תקופה טרם נבנו מחשבים דיגיטליים; חישוב ממוכן אז נעשה בשיטה האנאלוגית, בה גודל פיזיקלי כגון מרחק או מתח חשמלי ייצג ערך נומרי המתקבל או נוצר במהלך החישוב. אשר על כן תיאר טיורינג באורח תיאורטי בלבד מכונת חישוב פשוטה מאוד, מכונה הקרויה כעת על שמו **מכונת-טיורינג**, המסוגלת לבצע "כל חישוב שהוא". טיורינג לא ראה אפשרות או צורך בבניה בפועל של מכונה כזו, והוא אף הסביר כיצד ללא הפעלת כל מאמץ אינטלקואלי, יוכל כל אדם לחקות את פעולתה של מכונת טיורינג תוך שימוש בדף ועט.

אכן, מכונת טיורינג לא נבנתה מעולם, שכן, כפי שנראה בהמשך, איך בכך צורך או תועלת: מכונה כזו תהיה בהכרח איטית מאוד, וכתיבת תכניות בעבורה היא מסורבלת במיוחד. לעומת זאת, מכונות טיורינג וירטואליות קיימות לרוב, זאת)גם(כיוון שקל מאוד לבצע אמולציה של מכונת טיורינג ברוב שפות התכנות.

מכונת טיורינג[2] מקבלת קלט מסרט בודד (tape). מבנה הסרט שתיאר טיורינג מזכיר את הסרטים המגנטיים (magnetic tapes) שאמנם פותחו שנים ספורות לפני כן, אך טרם נוצלו כהתקן זיכרון של מחשב. סרט הוא רצועה אינסופית, אך חד כיוונית של תאים. כל אחד מתאים אלו עשוי להכיל שני סימבולים שונים. ניתן לסמן שני סימבולים אלו בספרות 0 ו-1, אולם מכונת טיורינג אינה משתמשת בסימבולים אלו כמספרים.

המכונה מייצרת את הפלט שלה על אותו הסרט עצמו עליו היא מקבלת את הקלט.

לבד מהסרט, מכונת טיורינג מצוידת בראש קריאה/כתיבה, הניצב בתחילת החישוב על התא הראשון בסרט, ואשר בכל נקודה במהלך החישוב על תא אחד מבין התאים שעל הסרט. למכונת טיורינג יש גם **מצב פנימי**, אותו ניתן לתאר במונחים של שפות תכנות מודרניות כמשתנה בודד, אשר יכול להכיל ערכים מוצבים ערכים הלקוחים מקבוצה סופית כלשהי. קיים גם ערך אחד מיוחד בקבוצה זו, הקובע את המצב הפנימי בתחילת פעולתה של המכונה.

קבוצת המצבים יכול להיות, למשל, קבוצת המספרים $1 \ldots N$ עבור ערך מתאים של N, וניתן גם להניח כי המספר 1 הוא המצב התחילי של המכונה. בכל זאת, מכונת טיורינג אינה יכולה לבצע חישובים עם מספרים אלו, אלא לבדוק את ערכם בלבד.

מכונת טיורינג מגדירה מעין שפת תכנות הקובעת כיצד תפעל המכונה, רשימת הנחיות המגדירה

1. מתי מסתיים החישוב, ומתי סיום זה יחשב כהצלחה
2. באלו תנאים על המכונה להזיז את הקריאה/כתיבה
3. אלו נתונים על המכונה לכתוב על הסרט, ובאילו נסיבות

מבנה רשימת ההנחיות הזו, או "שפת התכנות" של מכונות טיורינג הוא מנוונת מאוד מאוד. תכנית בשפה זו היא אוסף של פקודות פשוטות, הדומות במבנה שלהן לפקודות בשפת מכונה. ישנם חמישה סוגים של פקודות אותן מכירה מכונת טיורינג:

1. $\langle n \Rightarrow 3 \rangle$ פקודה הקובעת כי על המכונה לעצור את החישוב מיידית בכל עת שהמצב הפנימי הוא n; הפלט של המכונה ימצא אז על הסרט.
2. $\langle n \Rightarrow 7 \rangle$ פקודה הקובעת כי על המכונה לעצור והודיע כי החישוב נכשל בכל עת שהמצב הפנימי הוא n; לתכנו של הסרט אין משמעות כאשר החישוב נכשל.
3. $\langle n, h \Rightarrow -, n' \rangle$ פקודה הקובעת כי בכל עת שהמכונה נמצאת במצב n והסימבול עליו מצביע ראש הקריאה/כתיבה הוא h, $h \in \{ 0, l \}$ על המכונה לעבור למצב n', ולהזיז ראש זה צעד אחד שמאלה, כלומר, לכיוון תחילת הסרט.

[2] ישנן הגדרות רבות ושונות של מכונת טיורינג. אנו משתמשים כאן בהגדרה הפשוטה ביותר המתאימה לצרכי הדיון.

4. $\langle n, h \Rightarrow +, n' \rangle$ פקודה דומה לפקודה הקודמת, אלא שתזוזת הראש היא צעד אחד ימינה, כלומר לכיוון הצד האינסופי של הסרט.

5. $\langle n, h \Rightarrow h', n' \rangle$ פקודה הקובעת כי באותם תנאים, על המכונה להחליף את תוכן התא עליו מצביע הראש מהסימבול h לסימבול $h' \in \{0, 1\}$.

אוסף הפקודות הזה אינו מסודר. בכל איטרציה של החישוב בוחרת המכונה את הפקודה המתאימה למצב הפנימי הנוכחי שלה ולתוכן התא המצוי תחת ראש/הקריאה כתיבה, ומבצעת פקודה זו. ניתן להניח כי האוסף אינו מכיל פקודות סותרות, וכי אוסף הפקודות השונות מכסה את כל האפשרויות השונות בהן עשויה להיתקל המכונה)יש לכל היותר $2 \cdot N$ אפשרויות כאלו.(וכי החישוב אינו מזיז לעולם את ראש הקריאה/כתיבה שמאלה אל מעבר לתא הראשון שבסרט.

בהינתן פשטות פעולתה של מכונת טיורינג, ופשטותה של "שפת התכנות" שמאחוריה, מובן מאליו כי קל לכתוב, כמעט בכל שפת תכנות ובוודאי בפסקל או ב-C, תכנית אוניברסלית עבור מכונות טיורינג, כלומר תכנית המבצעת אמולציה של ריצתה של מכונת טיורינג. שפת תכנות היא טיורינג שלמה אם ניתן לכתוב בה תכנית אוניברסלית בעבור מכונת טיורינג. למעשה, קשה לחשוב על שפת תכנות שהיא כה פשוטה עד שלא ניתן יהיה לכתוב בה תכנית אוניברסלית שכזו.

ניתן להניח שבסופם של הנתונים הרשומים על הסרט שמקבלת מכונת טיורינג ישנו סימן מיוחד (למשל סידרה של תריסר מופעים של הסימבול 0). באופן דומה, נניח שטרם שמכונת טיורינג מסיימת את עבודתה, היא טורחת לכתוב סימן זה בסוף הסרט.

משמעו של הסימן המיוחד הזה הוא שבהינתן תכנית, רשימת הנחיות מסוימת, למכונת טיורינג, זו מחשבת פונקציה חלקית מסוימת מהקבוצה $\{0, 1\}^*$ אל עצמה. הפונקציה היא חלקית, לא רק משום שמכונת טיורינג עשויה לעצור את החישוב ולהודיע שהוא נכשל, אלא בעיקר, בגלל שמכונת טיורינג עשויה להכנס ללולאה אינסופית.

כל תכנית מגדירה פונקציה אחת כזו, אך כמובן יתכן שאותה פונקציה תוגדר על ידי מספר תכניות. טיורינג התעניין בקבוצת כל הפונקציות שאפשר לחשב באמצעות באמצעות המכונה שתכנן.

הזיהוי בין תכונת ה"אוניברסליות" של שפה ובין היותה טיורינג שלמה ראשיתו בהוכחה של טיורינג לפיה ניתן לחשב כל פונקציה רקורסיבית באמצעות תכנית מתאימה של מכונת טיורינג, ולהיפך, כל פונקציה אותה יכולה לחשב מכונת טיורינג היא פונקציה רקורסיבית. בניסוח אחר, קבוצת כל הפונקציות החלקיות אותן יכולה לחשב מכונת טיורינג, היא בדיוק קבוצת הפונקציות הרקורסיביות.

בד בבד עם עבודתו זו של טיורינג, פיתח אלונזו צ'רץ' (Church) מודל מתימטי אחר לתיאור האופן שבו נעשים חישובים. מודל זה קרוי תחשיב ה-λ, אותו התחשיב עליו תתבסס שפת ליספ, שלוש עשרה שנים מאוחר יותר. תחשיב ה-λ שונה בתכלית ממכונת טיורינג. הפעולה היסודית בתחשיב היא הפעלה של פונקציה על ערכים שאף הם עשוי שיהיו פונקציות. אולם המושגים של פונקציה והפעלה של פונקציה אינם קיימים מופיעים בהגדרתה של מכונת טיורינג. ומהצד האר, פעולות כגון קריאה וכתיבה של ערכים או שינוי ערכו משתנה מצב פנימי הן זרות בתכלית לתחשיב ה-λ.

משנתברר כי תחשיב ה-λ שקול למכונת טיורינג, במובן זה שניתן לכתוב בתחשיב תכנית אוניברסלית עבור "שפת התכנות" של מכונת טיורינג, ושניתן לתכנת מכונת טיורינג כך שתחקה את התחשיב, ובהתבסס על השקילות בין מכונות טיורינג ובין קבוצת הפונקציות הרקורסיביות, ניסחו צ'רץ' וטיורינג את הסברה הקרויה The Church Turing Thesis.

סברה זו, שאינה משפט מתימטי, נותנת מענה לשאלה "מהי כל שפת תכנות אחרת?" וזאת בקביעה כי **כל חישוב במודל חישובי "סביר", ניתן להעשות גם באמצעות מכונת טיורינג**. הסברה אינה קובעת מהו מודל חישובי סביר, ולכן היא פורשת כנפיה על מודלים של חישוב אשר טרם נודעו. בכל זאת, בחלוף שבעים שנה מניסוח הסברה, איש לא הצליח להפריך את הסברה באמצעות הצגת מודל חישובי אשר נראה "סביר" בעיני הקהילה המדעית, ולו גם מקצתה שבמקצתה.

הסברה של צ'רץ' וטיורינג מבהירה את הסיבה בגללה שפת תכנות שהיא טיורינג שלמה קרויה שפה אוניברסלית. מהסברה נובע עי שפה אשר יכולה לבצע אמולציה של מכונת טיורינג, יכולה לבצע אמולציה של כל מודל חישובי סביר אחר.[3]

3 אלגנטיות של שפת תכנות

ברור כי יש להבחין בין אוניברסליות ובין יעילות: על אף שליספ ופסקל הן שתיהן אוניברסליות, הרי תכניות הכתובות בליספ תדרושנה בדרך כלל יותר משאבים מתוכניות שקולות להן הכתובות בפסקל.

באופן דומה, יש להבחין גם בין אוניברסליות ובין נוחותו של המתכנת בכתיבת תכניות בשפה: כיוון ששפת ++C תומכת במבנים תכנותיים מתקדמים כגון מחלקות, העמסה של אופרטורים ופונקציות, אשר נעדרים משפת C נוכל לטעון ש-C חלשה יותר מ-++C. אך חולשה זו היא ענין של נוחותו של המתכנת. כיוון ש=C היא אוניברסלית, הרי לא זו בלבד שניתן, במאמץ מסוים, לכתוב בה כל תכנית שאפשר לכתוב בשפת ++C, אלא שניתן גם לחקות ב-C את המבנים המתקדמים של ++C, אף זאת במחיר של מאמץ תכנותי.

ישנם מדדים מספריים להשוואה בין שפות תכנות אוניברסליות מבחינת יעילות החישוב בהן. אולם, כיצד ניתן להשוות שפות תכנות מבחינת נוחות התכנות בהן, וכיצד נבחין בין נוחות, ובין טעם אישי של מתכנת זה או אחר?

לכאורה, מיטוב "נוחות התכנות" יכול להעשות באמצעות שפות תכנות האוספות אל תוכן כל מבנה תכנותי מתקדם אותו ניתן להעלות על הדעת, אלא ששפות אספניות נוטות לקרוס תחת כובד משקלן: נדרש מאמץ אנושי לא מבוטל כדי ללמוד כיצד לתכנת בהן, ונדרש מאמץ גדול אף יותר כדי לתכנן, לממש ולתמוך בכל המבנים הללו.

ג'ון מקארת'י התמודד עם השאלה הזו, כשהציג לראשונה את ליספ. הוא הראה אמנם שליספ היא שפה אוניברסלית, בכך שהיא יכולה לחשב כל פונקציה שיכולה לחשב מכונת טיורינג, אבל הוא רצה גם להוכיח באופן זה או אחר שליספ אלגנטית יותר ממכונת טיורינג. לשם כך, הוא הפעיל את השיקול הבא: אם שפה היא אוניברסלית, אז אפשר לכתוב ב-$\mathcal{L}$ תכנית אוניברסלית עבור $\mathcal{L}$ עצמה. כך, כיוון שמכונת טיורינג היא אוניברסלית, ניתן לכתוב עבורה תכנית אוניברסלית אחת, שתחקה את פעולתה של כל תכנית אחרת של מכונת טיורינג.

הקלט של התכנית האוניברסלית הזו יהיה התכנית האחרת, שאת פעולתה יש לחקות. לשם כך, תקודד רשימת הפקודות של התכנית באמצעות הסימבולים 0 ו-1, ותיכתב על הסרט של התכנית האוניברסלית. בהמשך הסרט, יש לכתוב את הקלט של התכנית המקורית.

באורח דומה, ניתן לכתוב בליספ תכנית אוניברסלית בעבור ליספ עצמה, והפונקציה eval **??** שכתב מקארת'י היא בדיוק פונקציה אוניברסלית עצמית כזו. בצדק טען מקארת'י שתכנית אוניברסלית עבור מכונת טיורינג תהיה ארוכה ומסורבלת. אמנם המשימה המוטלת על התכנית האוניברסלית של מכונת טיורינג היא פשוטה ביחס, אולם הפקודות הבסיסיות של מכונת טיורינג הן כה חלשות שיהיה צורך לכתוב רבות מאוד מהן כדי לבצע משימה זו.

לעומת זאת, התכנית האוניברסלית העצמית של ליספ נדרשת לבצע משימה קשה יותר: בגלל שהפעולות הבסיסיות של ליספ אותם היא נדרשת לחקות, מורכבות יותר מהפקודות הפשוטות של מכונת טיורינג. אולם, לרשות התכנית בליספ עומדים כלים חזקים יותר כדי לעשות זאת. העובדה שניתן לממש את eval כל כך בקלות, כך טען מקארת'י, היא עדות לאלגנטיות של ליספ.

נכליל את השיקול הזה, ונשתמש באורך התכנית האוניברסלית העצמית של שפת תכנות כמדד לאלגנטיות של השפה: אם השפה מורכבת ומתוחכמת, הרי, מחד, קל יותר לכתוב את התכנית הזו, אך מאידך, על התכנית

[3] המכורים למשחק המחשב Minecraft ישמחו לשמוע כי היה מי שבנה מכונת טיורינג באמצעות redstone. המשחק לכן אינו רק מהנה, הוא גם אוניברסלי!

לעסוק בכל המורכבות והתיחכום הזה. לחילופין, שפה שהיא פשוטה מאוד כמו מכונת טיורינג (וגם כמו שפת ה-batch של מערכת ההפעלה DOS) שהוזכרה מעלה, היא קלה יותר לעיבוד ולביצוע, אבל הפשטות של השפה מהווה אבן נגף בבואנו להשתמש בה כדי לכתוב את התכנית האוניברסלית העצמית.

בניסוח אחר: ככל שהתכנית אוניברסלית העצמית של שפת תכנות מסוימת קצרה יותר, כך נראה שהבחירה והארגון של המבנים התכנותיים בשפה "נכונה" יותר: בחירה שהיא מספיק חזקה כדי לממש את עצמה בתמציתיות.ככל שהשפה מציעה מבנים מתקדמים יותר, כך קל יותר ו"נוח" יותר לממש בה תכנית אוניברסלית עצמית. אולם, ככל שיש יותר מבנים כאלו, וככל שהם מתקדמים יותר, כך נדרשת התכנית האוניברסלית העצמית לבצע יותר.

הנה מספר נתונים על אורכן של תכניות אוניברסליות עצמית.

1. מהדר לשפת C הכתוב בשפה C עצמה, דורש כמה עשרות אלפי שורות.
2. שפת C+++ העשירה הרבה יותר מ-C, אינה מועילה לקיצור המהדר. המהדר של gcc מתפרס על פני כשבעה מיליון שורות קוד.
3. המהדר הראשון לשפת פסקל, אשר נכתב בשפת פסקל דרש כשבעת אלפים ומאתיים שורות.
4. כפי שראינו כאן, הפונקציה האוניברסלית `eval` בעבור מיני-ליספ,
5. לעומת זאת, מפרש בסיסי לשפת התכנות פרולוג הכתוב בפרולוג יכול להכתב בשורה אחת בלבד, ומפרש מתוחכם, המאפשר למשל מעקב אחרי החישוב, לא ידרוש בדרך כלל יותר מעשר שורות.

4 מימוש שפת תכנות באמצעות עצמה

מתברר שמקובל מאוד לכתוב את המהדר של השפה בשפה עצמה. כך למשל, המהדר של שפת C כתוב בִּשְׂפַת C. המהדר של שפת גָ'וָה כתוב בִּשְׂפַת גָ'וָה, וכו'. כמובן שהדבר מעורר קושי: כי אם המהדר עבור שפה מסוימת כתוב באותה שפה, הרי כיצד הודר המהדר? התשובה הפשוטה היא שהמהדר הידר את עצמו, וכך הם בדרך כלל פני הדברים. אלא, שהדבר מוביל לרקורסיה אינסופית.

בכל זאת, בשפות תכנות, ניתן לבצע תהליך של bootstrapping שבאמצעותו ניתן לפתח מהדר המסוגל להדר את עצמו. התהליך דומה מאוד למה שהיה עושה נפח עני שברשותו ברזל, אך לא כסף לרכישת צבת. נפח כזה היה משתמש בכבשנו באופן איטרטיבי, כאשר בכל פעם הוא היה משתמש בגוש הברזל הדומה ביותר לצבת שיש ברשותו, כדי ליצר קירוב טוב יותר לצבת.

תהליך ה-bootstrapping (שעדות לו נמצאת ב**??**) מציע את השיטה האיטרטיבית הבאה למימוש שפת תכנות חדשה באמצעות עצמה:

ראשית יש להגדיר תת שפה ראשונית של השפה החדשה אותם רוצים לממש. כך עשינו כשהגדרנו את מיני-ליספ. לאחר מכן, יש לכתוב את המהדר לתת השפה הזו באמצעות עצמה. את המהדר הראשון של שפת המכונה ניתן לתרגם ידנית לשפת מכונה. אפשרות אחרת היא לתקן ולשנות מהדר של שפת תכנות קיימת, כך שיתמוך בתת השפה הראשונית. כך נעשה במימוש הראשוני של פסקל.

לאחר ההידור ההתחלתי של המהדר הראשון, נהדר אותו באמצעות עצמו. אחר כך, המהדר הראשון יורחב כך שיתמוך בתת שפה שניה, מעט גדולה יותר. גם מהדר זה יוכל להדר את עצמו, ולאחר שכך יקרה, ניתן לשפר את המהדר תוך שימוש בתכונות של תת-השפה השניה. כדי ליצור מהדר לשפה החדשה שתומך בכל מרכיביה, יש להמשיך ולחזור ככל שידרש על תהליך ה-bootstrapping: הרחבת המהדר כך שיתמוך בתכונות נוספות של השפה, ושיכתוב המהדר כך שישתמש בתכונות אלו.