פרויקט סופי בקורס סינתזה של תוכנה 236347:

מגישים: שלומית הרוש 209452630 shlomit.har@campus.technion.ac.il
אלון מור 319040655 alonm16@campus.technion.ac.il
נועם דוד חיון 207060633 noam.hayun@campus.technion.ac.il

בפרויקט יצרנו synthesizer בשפת python, אשר מקבל כקלט כללי גזירה לתיאור השפה שבמרחבה הוא מחפש תוכניות וכן specification על ידי רשימת דוגמאות קלט פלט (או על ידי symbolic examples כפי שיתואר בהמשך) החיפוש נעשה על ידי אלגוריתם bottom up כפי שנלמד בהרצאה.
פרמטרים נוספים הינם מגבלת זמן ומגבלת עומק (ערכי ברירת המחדל שלהם הינם 70 שניות ועומק 5), וכן אפשרות להוסיף specification with symbolic examples ופונקציה ליצירת קלטים לפונקציות lambda אשר ישמשו למימוש OE גם לפונקציות lambda.
בנוסף המשתמש יכול לאפשר לsynthesizer לייצר תוכניות שיטפלו בקלטים מסוימים באופן שונה (condition abduction for corner cases), על המשתמש לספק במקרה זה גם כללי גזירה לגזירת התנאי המבדיל בין הקלטים.
על הרחבות אלו ואופטימיזציות נוספות נרחיב בהמשך.

הקבצים בפרויקט:

- **synthesizer.py** - קובץ זה מכיל מימוש של שתי מחלקות מרכזיות בפרויקט, Program ו- Synthesizer
  Program מייצגת תוכנית, לכל תוכנית שמרנו את הקוד שלה במחרוזת בשדה code, שדה outputs שהינו רשימה של הפלטים שלה על הקלטים שניתנו ב specification וכן את העומק שבו התגלתה התוכנית.
  Synthesizer המחלקה המרכזית המסנתזת תוכנית בהתאם לספציפיקציה.
- **funcs.py** - קובץ זה מכיל פונקציות שניתן לקרוא להן בכללי הגזירה.
- **tests.py** - קובץ זה מכיל את הדקדוקים והbenchmarks איתם בדקנו את הפרויקט, יורחב בהמשך.

אופטימזציות:

- Observational Equivalence - ביצענו זאת על ידי שמירת השדה seen_outputs במחלקה **Synthesizer**.
  שדה זה הינו מילון הממפה משתנה גזירה לset של מחרוזות המייצגות פלטים של תוכניות שכבר נראו, וכאשר משתנה גוזר תוכנית טרמינלית חדשה, אנו מחשבים מחרוזת של הoutputs שלה (על הקלטים בspec) ובודקים האם המחרוזת נמצאת בset של משתנה זה, במידה וכן אנו מתעלמים מתוכנית זו, אחרת אנו מוסיפים את הפלט שלה לset ומתחשבים בה.
  החיפוש בset בpython יעיל מאוד (**O(1)** בממוצע) ובנוסף החלוקה לset לכל משתנה דקדוק מאפשרת לנו לבצע OE לכל משתנה דקדוק בנפרד. נציין שאין צורך לבצע OE בין תוכניות שנגזרו ע"י משתנים שונים, מפני שיש להם תפקידים שונים ויכולים להמצא בכללי גזירה שונים, ולכן גם אם נמצאו תוכניות שקולות הנגזרות על ידי משתנים שונים צריך להשאיר את שתיהן.
- נפתח תוכניות מכלל גזירה רק אם יש לפחות משתנה אחד מאגף ימין שגזר תוכנית באיטרציה הקודמת (אחרת כל התוכניות שנפתח בשלב הנוכחי כבר פותחו בשלב הקודם ונרצה להימנע מפיתוח שלהן בשנית), אופטימזציה זו ממומשת על ידי שמירת מיפוי vars_depth במחלקה **Synthesizer** הממפה משתנה גזירה לעומק המקסימלי שבו הוא גזר תוכנית.

- בנוסף כפי שציינו לכל תוכנית יש שדה Depth המציין את העומק בו היא נגזרה, כאשר אנו גוזרים תוכנית חדשה, חייב שלפחות אחת מהתוכניות מהן היא מורכבת היא תוכנית מהאיטרציה הקודמת. אחרת, אם כל התוכניות ממנה היא מורכבת הינם מאיטרציות ישנות יותר- נתעלם מתוכנית זו.

Something Extra:

1. **Symbolic Examples -** רצינו לספק נוחות למשתמש ולכן אפשרנו לו לתת ספציפיקציה לתוכניות שהוא מעוניין לסנתז על ידי 2 דרכים שונות או שילוב של שתיהן.
הדרך הראשונה הינה לתת רשימה של tuples המייצגים קלט ופלט של התוכנית.
בדרך השניה המשתמש נותן רשימה של tuples, כל tuple הינו מהצורה $("input",\ "output")$
כאשר input וoutput הינם קלט ופלט **שיכולים להכיל משתנים**: a,b.
לדוגמא: $("[a,b]",\ "a+b"),\ ("[5,a,b]",\ "a"),\ ("a",\ "a*A")$
ישנו אילוץ שהמשתנים a,b הינם מטיפוס int.
מכל אילוץ "סימבולי" בודד שנתן המשתמש יצרנו רשימת קלטים ופלטים קונקרטיים.
אם באילוץ היה משתנה בודד, יצרנו 20 קלטים ופלטים קונקרטיים ואילו היו שני משתנים, יצרנו 10 השמות לכל משתנה וסה"כ 100 קלטים ופלטים קונקרטיים.
בחנו שימוש בתוספת זו בטסטים המתחילים בtest_sym_ex
2. **סינתזה של פונקציות lambda וכן Observational Equivalence ביניהן -** סינתוז הפונקציה עצמה מתאפשר על ידי הדקדוק המסופק (כפי שניתן לראות בקובץ tests.py בדקדוקים של lambda funcs). רצינו לממש גם OE בין פונקציות הlambda המסונתזות, לצורך כך הוספנו לconstructor של Synthesizer פרמטר נוסף, אופציונלי - פונקציה המייצרת מופע של קלט אפשרי לפונקציית למדא. (למשל אם פונקציות הלמדא מקבלות int אז פונקציה לדוגמא יכולה להיות למשל יצירת int רנדומלי בטווח מסוים), בעזרת פונקציה זו אנו מייצרים 30 קלטים אפשריים, ואיתם נבדוק שקילויות של פונקציות הלמדא, כלומר- שתי פונקציות למדא יחשבו שקולות אם הפלטים שלהן על כל 30 הקלטים שנוצרו זהים. בדיקת השקילות מבוצעות על ידי בדיקה בset באופן דומה להסבר על OE בין תוכניות. כאשר סינתזנו פונקציית למדה, נחשב את הפלטים שלה על 30 הקלטים, ובמידה וכבר צפינו בפלטים אלו נתעלם ממנה. בכך אנו מבטלים פונקציות שקולות.
3. **Condition Abduction For Corner Cases -** המשתמש יכול להשתמש באפשרות זו על ידי קריאה לפונקציה find_solution_with_condition_abduction(condition_grammar) של המחלקה Synthesizer.
מימשנו זאת באופן הבא: ראשית הרצנו את find_solution לבדוק האם קיימת תוכנית מתאימה ללא צורך לחלוקה למקרים, במידה וכן- החזרנו את התוכנית המתאימה.
אחרת- חיפשנו מבין התוכניות שכבר ייצרנו את התוכנית שתאמה למספר מקסימלי של זוגות קלט ופלט מתוך הספציפיקציה, נסמן תוכנית זו בprog1. כעת, לקחנו את הקלטים והפלטים ש-prog1 טעתה בהם ומהם יצרנו תת ספציפיקציה , סינתזנו תוכנית עבור תת הספציפיקציה נסמנה prog2.
בשלב הסופי נותר לנו לסנתז את התנאי אשר יבחין לנו בין הקלטים שמתאימים לprog1 לאלו המתאימים לprog2. עבור הסנתוז של התנאי המשתמש מספק דקדוק כפרמטר לפונקציה-condition_grammar, וכן אנו יצרנו ספציפיקציה שנותנת true לקלטים שprog1 הצליחה עבורם וfalse לשאר, את התנאי סינתזו עם ספציפיקציה זו ועם הדקדוק המסופק, נסמן את התנאי שסונתז בcond. כמובן שאם לא נמצא prog2 או cond העונים לדרישות החזרנו כי אין תוכנית מתאימה.
במידה וכן נמצאו, התוכנית הסופית שהחזרנו הינה: $prog1\ if\ cond\ else\ prog2$.

Testing

על מנת לבדוק את אופן פעולת ה- synthesizer , כתבנו 40 בדיקות (טסטים). את סיכום התוצאות של הטסטים ניתן למצוא תחת הקובץ synthesizer_tests_report.csv, בו כל שורה מציינת עבור טסט יחיד את שמו, האם הsynthesizer הצליח למצוא תוכנית מתאימה וכן משך הזמן שעבר עד שנמצאה התוכנית.
בנוסף עבור כל טסט נפלט קובץ שמציין את התוכנית שנמצאה עבורה, או הערה שמציינת שלא נמצאה תוכנית מתאימה בתיקיה tests_results.

- **ניתן להריץ את הבדיקות מחדש ע"י הרצת הקובץ tests.py**

הדקדוקים שנבדקו:
arithmetic_grammar - דקדוק שנועד לבדוק פעולות אריתמטיות.

string_grammar - דקדוק שנועד לבדוק פעולות על מחרוזות, במסגרת הטסטים שנבדקו בדקנו השלמת מחרוזת, הכפלת מחרוזת, שרשור של מחרוזת lower למחרוזת UPPER בהינתן מחרוזת בצורה כלשהי ועוד.

list_grammar1, **list_grammar2** - דקדוקים שנועדו לפקוד פעולות על רשימות, הדקדוק מכיל בתוכו פקודות ליצירת רשימה חדשה, שרשור רשימות, reverse, sort, find ועוד.

lambda_grammar1, **lambda_grammar2** - שני דקדוקים שנועדו לבדוק את אופן פעולת הsynthesizer בשימוש בפונקציות lambda וכן את השיפור השני בתואר לעיל.

condition_abduction_grammar, **boolean_grammar** - שני דקדוקים אלו נכתבו על מנת לבדוק את השיפור השלישי שתואר לעיל.

בנוסף בחנו את השיפור הראשון שתואר לעיל בטסטים ששמם מתחיל ב: test_sym_ex