**תורת הקומפילציה (61304)**

מרצה: ד"ר מרק טרכטנברוט

סמסטר א', תשע"ט.

מועד א', **1.04.19**

* משך הבחינה: 3 שעות.
* חומר עזר: **דף עזר** המצורף לשאלון זה (עמודים 4-5).
* במבחן **6** שאלות**. יש לענות על כולן.**
* ניתן לצבור **עד *110*** נקודות, אך **ציון הבחינה לא יעלה על 100**.
* **תשובות ללא נימוק לא תתקבלנה!**
* יש להקפיד על כתב ברור וקריא. ע"פ התקנון לא ניתן לכתוב בחינה בעפרון, אלא בעט בלבד.
* יש להתחיל כל תשובה בעמוד נפרד.

**שאלה 1** (15 נקודות)

יש לרשום ביטוי רגולרי מורחב המתאר כתובות של דואר אלקטרוני, לפי דרישות הבאות:

* צורה כללית של הכתובת :

[USER\_NAME@DOMAIN.SUFFIX](mailto:USER_NAME@DOMAIN.SUFFIX)

* USER\_NAME מורכב מאותיות (קטנות וגדולות) וספרות, ויכול להכיל גם שני תווים מיוחדים: קו-תחתון ונקודה, לפי חוקים הבאים:
  + - חייב להתחיל באות
    - הנקודה יכולה להופיע רק פעם אחת, ואילו קו-תחתון יכול להופיע מספר פעמים
    - תווים מיוחדים לא יכולים להופיע בסוף של USER\_NAME; כמו כן הם לא יכולים להופיע ברצף אחד אחרי השני.
* DOMAIN מורכב מאותיות (קטנות וגדולות) וספרות, ויכול להכיל תו מקף בודד
* SUFFIX יכול להיות אחד מהבאים: com, co.il, ac.uk, de
* הפתרון שלכם חייב להשתמש, בין השאר, בכל האופרטורים הבאים לצורך בניית הביטוי הרגולרי המורחב:
* [ ...] : טווח
* ?R : אפס או אחד מופעים של הביטוי R
* "S" : מחרוזת S מופיע בקלט כפי שהיא כתובה בין הגרשיים.

**דוגמאות של כתובות חוקיות:**

mark\_tr2019@walla.co.il

john.hopkins@oxford-uni1.ac.uk

Long\_user\_name.Family@siemens.de

**דוגמאות של כתובות לא חוקיות:**

Moshe\_Cohen@hit.ac.il

david.avi.paz@gmail.com

user@new-high-school.ac.uk

**פתרון**

LETTER [A-Za-z]

LETTER\_OR\_DIGIT [A-Za-z0-9]

UDERSCORE\_PARTS (“\_”?{LETTER\_OR\_DIGIT})\*

/\* reminder: “\_” can appear any amount of times \*/

DOT\_PART “.”{LETTER\_OR\_DIGIT}{UDERSCORE\_PARTS}

/\* reminder: “.” can appear only once \*/

NO\_DOT\_NAME {LETTER}{UDERSCORE\_PARTS}

USER\_NAME {NO\_DOT\_NAME}{DOT\_PART}?

MAKAF\_PART “-“{LETTER\_OR\_DIGIT}+

/\* reminder: “-“ can appear only once \*/

DOMAIN {LETTER}{LETTER\_OR\_DIGIT}\*{MAKAF\_PART}?

SUFFIX “com” | “co.il” | “ac.uk” | “de”

ADDRESS {USER\_NAME}”@”{DOMAIN}”.”{SUFFIX}

**שאלה 2** (15 נקודות)

בצע סילוק של גורמים שמאליים משותפים וסילוק של רקורסיה שמאלית לדקדוק הבא:

EXP 🡪 num | id | ADDR | ARRAY\_ELM

ARRAY\_ELM 🡪 id[EXP]

ADDR 🡪 BASIC\_ADDR | EXP\_ADDR

BASIC\_ADDR 🡪 &id

EXP\_ADDR 🡪 ADDR+num | ADDR\*num

**שימו לב** שיש להתייחס גם למופעים המפורשים (direct) וגם למופעים עקיפים (indirect) של התופעות המדוברות בדקדוק.

פתרון

1. סילוק של גורמים שמאליים משותפים.

אחרי הצבת ההגדרה של ARRAY\_ELM נקבל:

EXP 🡪 num | id | ADDR | id[EXP]

וכך נחשפים גורמים שמאליי שהם משותפים בעקיפין. הסילוק מתבצע בעקבות זה:

EXP 🡪 num | ADDR | id E

E 🡪 ε | [EXP]

1. סילוק שלרקורסיה שמאלית.

בהגדרה של ADDR, נבצע הצבה במקום EXP\_ADDR כדי לחשוף רקורסיה שמאלית הקיימת בדקדוק בעקיפין; נקבל:

ADDR 🡪 BASIC\_ADDR | ADDR+num | ADDR\*num

כעת מבצעים את הסילוק:

ADDR 🡪 BASIC\_ADDR A

A 🡪 ε | +num A | \*num A

**שאלה 3** (20 נקודות)

נתון דקדוק:

S → aAc | bSBe | Aba

A → BS | dS

B → cB| ε

בצע חישוב של התכונות Nullable, First , Follow עבור משתני הדקדוק.

**שימו לב**: יש להציג את כל שלבי החישוב, ולא רק את התוצאה הסופית.

### פתרון

א)

חישוב של תכונת Nullable:

רק משתנה B אפיס; בכל כללי גזירה עבור משתנים S,A מופיע לפחות אסימון אחד, ולכן הם לא אפיסים.

חישוב של קבוצות First עבור משתני הדקדוק:

אתחול: כל קבוצות ריקות.

*איתרציה 1:*

First(S) = First(S) U {a} U {b} U First(A) = {a,b}

First(A) = First(A) U First(B) U First(S) /\* B is nullable \*/ U {d} = {a,b,d}

First(B) = First(B) U {c} = {c}

*איתרציה 2:*

First(S) = First(S) U First(A) = {a,b} U {a,b,d} = {a,b,d}

First(A) = First(A) U First(B) U First(S) /\* B is nullable \*/ = {a,b,d} U {c} U {a,b} = {a,b,c,d}

*איתרציה 3:*

First(S) = First(S) U First(A) = {a,b,d} U {a,b,c,d} = {a,b,c,d}

First(A) = First(A) U First(B) U First(S) /\* B is nullable \*/ =

{a,b,c,d} U {c} U {a,b,c,d} = {a,b,c,d}

*איתרציה 4:*

אין שינויים.

חישוב של קבוצות Follow עבור משתני הדקדוק:

אתחול: Follow(S) = EOF, שאר הקבוצות - ריקות.

*איתרציה 1:*

Follow(A) = Follow(A) U {c} = {c}

Follow(S) = Follow(S) U First(B) U {e} /\* B is nullable \*/ = {EOF, c, e}

Follow(B) = Follow(B) U {e} = {e}

Follow(A) = Follow(A) U {b} = {b,c}

Follow(B) = Follow(B) U First(S) = {a,b,c,d,e}

Follow(S) = Follow(S) U Follow(A) = {EOF, b,c,e}

Follow(B) = Follow(B) U Follow(B) = {a,b,c,d,e}

*איתרציה 2:*

אין שינויים.

סיכום:

|  |  |  |  |
| --- | --- | --- | --- |
| Follow | First | Nullable |  |
| EOF,b,c,e | a,b,c,d | no | S |
| b,c | a,b,c,d | no | A |
| a,b,c,d,e | c | yes | B |

**שאלה 4** (כל סעיף - 3 נקודות, סה''כ 18 נקודות)

לפניך סוגים שונים של שגיאות. עבור כל אחת מהן יש להסביר האם קומפיילר מסוגל לגלות אותה, ואם כן אז באיזה שלב של הקומפילציה (אם קיימות כמה אפשרויות, יש להסביר במה זה תלוי) :

1. שימוש ב- idשמוגדר כ exception בתוך ביטוי אריתמטי , כגון x+1

שגיאה סמנטית: אי-אפשר לבצע פעולות אריתמטיות על אוביקט שהוגדר כ-exception

1. שימוש ב-id שהוגדר כמשתנה (ולא כקבוע) כדי לציין גודל של מערך ; למשל int a[my\_variable]

אם המשתנו אינו מאותחל – אז שגיאה סמנטית.

יש שפות שבהן בהגדרת גודל המערך יכול להופיע רק קבוע. גם במקרה זה – שגיאה סמנטית.

1. שימוש בסוגריים מרובעים [ ] במקום סוגריים רגילים ( ) בכותרת של לולאה מסוג for ; למשל:

for [i=0; i<10, i++] {a[i+1] = a[i]\*2};

שגיאה תחבירית: אחרי מילה שמורה for חייב לבוא אסימון ) ולא ] . למעשה, הכותרת כולה של לולאה מסוג for חייבת להופיע בסוגריים מהסוג ( ) , ולא מהסוג [ ] .

1. אופרציה לא חוקית, למשל x+++ ;

שגיאה תחבירית. מבחינה לקסיקלית הכל תקין: מנתח לקסיקלי מזהה סדרת אסימונים הבאים (בהתאם למדיניות זיהוי של אסימון עם לקסמה ארוכה ביותר אפשרית):

id op\_plusplus op\_plus semicolon

אבל מבחינה תחבירית, אחרי אסימון op\_plusplus לא מצפים לראות את op\_plus , אלא אסימונים כמו semicolon או close\_parenthesis , זאת אומרת ; או (

1. פניה לאיבר במערך, כאשר ערך של אינדקס עולה על גודל המערך.

אם אינדקס הוא מספר מפורש העולה על גודל המערך אז קומפיילר יגלה את זה בזמן ניתוח סמנטי. אחרת זאת שגיאה של זמן ריצה.

1. שימוש בסוגריים מרובעים בקריאה לפונקציה , למשל max[a,b]

שגיאה סמנטית. מבחינה תחבירית זה תקין כי כך יכולה להיראות פניה לאיבר במערך. אבל אחרי שקומפיילר מגלה (בטבלת סמלים) ש max מוגדר כפונקציה, הוא מוציא הודעת שגיאה סמנטית.

**שאלה 5**  (15 נקודות)

עבור הדקדוק משאלה 3, כתבו פונקציות parse לביצוע ניתוח תחבירי לפי שיטת Recursive Descent Parsing. פונקציות אלה חייבות להכיל טיפול בהתאוששות משגיאות תחביר.

לצורך כתיבת הפונקציות, יש להשתמש בתוצאות החישוב של קבוצות First, Follow .

**הערה:** ניתן להשתמש בפונקציות עזר nextToken(), backToken(), match().

**פתרון**

הדקדוק הנתון (עם מספור של כללי גזירה):

1 2 3

S → aAc | bSBe | Aba

4 5

A → BS | dS

6 7

B → cB | ε

void parse\_S()

{cur\_token = next\_token();

switch (cur\_token 🡪 kind) {

/\* S is not nullable; hence need only cases for tokens from First(S) \*/

case a: {print(“Rule 1”); parse\_A(); match(c); break}

case b: {print(“Rule 2”); parse\_S(); parse\_B(); match(e); break}

case c: case d: {print(“Rule 3”); cur\_token = back\_token();

parse\_A(); match(b); match(a); break}

default: {print(ERROR);

while cur\_token is not in Follow(S) && cur\_token !=EOF do

cur\_token = next\_token();

cur\_token = back\_token()}

}

}

void parse\_A()

{cur\_token = next\_token();

switch (cur\_token 🡪 kind) {

/\* A is not nullable; hence need only cases for tokens from First(A) \*/

case a: case b: case c:

/\* taking into account that B is nullable \*/

{print(“Rule 4”); cur\_token = back\_token(); parse\_B(); parse\_S(); break}

case d: {print(“Rule 5”); parse\_S(); break}

default: {print(ERROR);

while cur\_token is not in Follow(A) && cur\_token !=EOF do

cur\_token = next\_token();

cur\_token = back\_token()}

}

}

void parse\_B()

{cur\_token = next\_token();

switch (cur\_token 🡪 kind) {

/\* B is nullable; start with cases for tokens from First(B) \*/

case c: {print(“Rule 6”); parse\_B(); break}

/\* and now continue with cases for tokens from Follow(B) \*/

case a: case b: case c: case d: case e:

{print(“Rule 7”); cur\_token = back\_token(); break}

/\* Note the contradicting cases for token c: this is because c belongs to both

First(B) and Follow(B).

The meaning of this contradiction: look-ahead of one token only is not enough for

this grammar. In case when the current token is c, parser must check at least one

more token in order to make a correct choice \*/

default: {print(ERROR);

while cur\_token is not in Follow(B) && cur\_token !=EOF do

cur\_token = next\_token();

cur\_token = back\_token()}

}

}

**שאלה 6** (א' - 17 נקודות, ב' - 2 נקודות, ג' – 2 נקודות, ד' - 6 נקודות ; סה''כ 27 נקודות)

נתון דקדוק להגדרת תהליכים המבצעים במקביל פעולות קריאה וכתיבה (**write** ו- **read**):

#### S 🡪 parbegin P parend

#### S 🡪 read

#### S 🡪 write

P 🡪 P **||** S

P 🡪 S

**הערה:** || הינו אסימון אחד (אופרציה "ביצוע במקביל")

יש להבדיל בין אסימון זה לבין אופרצית שרשור מחרוזות ++ שניתן להשתמש בה בפתרון השאלה.

1. בנה הגדרה מונחית תחביר (הכוללת את כללי גזירה הנתונים ופעולות סמנטיות מתאימות) אשר הופכת מחרוזת קלט למחרוזת המתקבלת ממנה ע"י החלפות הבאות:
2. כל פעולות **write** מופיעות לפני כל פעולות **read**
3. בפלט כל הפעלות מוקפות על-ידי זוג אחד של "סוגריים" **parbegin-parend**

**לדוגמא**: אם קלט הוא:

**parbegin write || parbegin read || write parend parend**

אז הפלט הוא:

**parbegin write || write || read parend**

1. הגדר את כל התכונות הסמנטיות בהן נעשה שימוש בתשובתך לסעיף א' (כולל הסבר קצר מהו תפקידן), וסווג אותן (נוצרות / נורשות).
2. הסבר מהו הסדר המדוייק של ביצוע הפעולות בתשובתך לסעיף א'.
3. על בסיס ההגדרה שלך לסעיף א' , בנה עץ גזירה המורחב (המכיל את ערכי התכונות בכל צומת) עבור מחרוזת קלט הבאה**: parbegin write || read || write parend**

**שימו לב:**

* ניתן להשתמש בתכונות סמנטיות בלבד (בפרט, לא ניתן להשתמש במשתנים גלובלים ו/או מבני נתונים ו/או קריאה לפונקציות כלשהן, מלבד פונקצית הדפסה).
* קריאה לפונקצית הדפסה יכולה להופיע אך ורק במשתנה התחלתי של הדקדוק. אסור לשנות את הדקדוק.

**פתרון**

נרחיב את הדקדוק הנתון על ידי הוספה של משתנה התחלתי חדש S’ (הסיבה: משתנה התחלתי המקורי מופיע גם בצד ימין של כללי גזירה):

#### S’ 🡪 S S 🡪 parbegin P parend

#### S 🡪 read

#### S 🡪 write

P 🡪 P **||** S

P 🡪 S

הפתרון מתבסס על רעיון פשוט הבא:

1. במעבר על מחרוזת קלט, אוספים את כל המופעים של פעולה write במחרוזת נפרדת, ואוספים גם את כל פעולות read במחרוזת נפרדת
2. יוצרים פלט על ידי שרשור של שתי המחרוזות הנ''ל בסדר הנדרש (כדי להבטיח ש בתוצאה כל פועולות write יופיעו לפני כל פעולות read ) והוספה מסביב של זוג "סוגריים" parbegin-parend .

תכונות סמנטיות הרלוונטיות:

S.str\_write - כאן נאספות כל פעולות write המופיעות במחרוזת שנגזרת ממשתנה S

P.str\_write - דומה, עבור משתנה P

S.str\_read - כאן נאספות כל פעולות read המופיעות במחרוזת שנגזרת ממשתנה S

P.str\_read - דומה, עבור משתנה P

S’.str – מחרוזת זאת מכילה בתוכה את כל הפעולות שמופיעות בקלט, בסדר הנדרש

כל התכונות הנ''ל הן נוצרות, ולכן קודם מתבצעת גזירה, ואחרי זה – ביצוע של פעולות סמנטיות.

להלן הגדרה מונחית תחביר העונה על דרישות השאלה:

#### S’ 🡪 S {S’.str = if (S.read\_str != “” && S.write\_str != “”

then S.write\_str ++ “||” ++ S.read\_str

else S.write\_str ++ S.read\_str

#### print(“parbegin” ++ S’.result ++ “parend”)} S 🡪 parbegin P parend {S.read\_str = P.read\_str; S.write\_str = P.write\_str}

#### S 🡪 read {S.read\_str = “read”; S.write\_str = “”}

#### S 🡪 write {S.read\_str = “”; S.write\_str = “write”}

P 🡪 P1 **||** S {P.read\_str = if (P1.read\_str != “” && S.read\_str != “”)

then P1.read\_str ++ “||” ++ S.read\_str

else P1.read\_str ++ S.read\_str

P.write\_str = if (P1.write\_str != “” && S. write \_str != “”)

then P1. write \_str ++ “||” ++ S. write \_str

else P1. write \_str ++ S. write \_str

P 🡪 S {P.read\_str = S.read\_str; P.write\_str = S.write\_str}

עץ מורחב עבור מחרוזת קלט הנתונה בשאלה:

S’.str = “write || write || read” S’ print(“parbegin write || write || read parend”)

S S. write\_str = “write || write”

S. read\_str = “read”

parbegin P parend P. write\_str = “write || write”

P. read\_str = “read”

P. write\_str = “write” S. write\_str = “write”

P. read\_str = “read” P || S S. read\_str = “”

P. write\_str = “write” P || S S. write\_str = “” write

P. read\_str = “” S. read\_str = “read”

S.write\_str = “write” S read

S.read\_str = “”

write