# פתרון תרגיל בית מס' 5

## שאלה 1:

א. בשביל לדאוג בתחביר שפקודת break תהיה רק בתוך מבני בקרה מורכבים יש לעשות מספר דברים:

1. יש להוסיף משתנה דקדוק חדש (נקרא לו 'S) ולשכפל את כל כללי הגזירה של S תוך החלפת מופעים באגף ימין של המשתנה S במשתנה 'S.
2. להוסיף כלל $S' \rightarrow$ break.
3. בכללים של S הגוזרים מבנה בקרה מורכב יש להחליף במשתנה 'S את המופעים של המשתנה S שבמבנה המורכב.

לדוגמה:

בעקבות הכלל

$S \rightarrow$ if E then $S_1$ else $S_2$

יתווסף הכלל

$S' \rightarrow$ if E then $S'_1$ else $S'_2$

ואילו הכלל

$S \rightarrow$ while ( E ) $S_1$

יוחלף בצמד הכללים

$S \rightarrow$ while ( E ) $S'_1$
$S' \rightarrow$ while ( E ) $S'_1$

שיטה זו אינה יעילה כי היא דורשת הכפלת הדקדוק.

ב. לכל מבנה בקרה נגדיר תכונה נורשת בוליאנית שמשמעותה תהיה "האם אני נמצא בתוך מבנה בקרה מורכב?". מבנה בקרה מורכב תמיד יאתחל עבור בניו את התכונה הזו לערך אמת ואילו מבנה בקרה פשוט יוריש לבניו את ערך התכונה שלו.
בכלל $S \rightarrow$ break נוודא שלתכונה ערך אמת ואם לא אז נודיע על שגיאה.

ג. לכל מבנה בקרה נגדיר תכונה נוצרת בוליאנית שמשמעותה תהיה "האם אני מכיל פקודת break שלא עטופה במבנה בקרה מורכב?".
עבור הכלל $S \rightarrow$ break תכונה זו תקבל ערך אמת.
עבור מבנה בקרה פשוט, תכונה זו תקבל את איחוד תכונות הבנים.
עבור מבנה בקרה מורכב, תכונה זו תקבל ערך שקר (אם חלקים מהפקודה פשוטים וחלקים מורכבים – תקבל איחוד תכונות הבנים שבחלקים הפשוטים).
בכלל שגוזר את ה- S הראשי יש לוודא שערך התכונה הוא ערך שקר, ואם לא אז להודיע על שגיאה.

ד. נוסיף ל- S תכונה נוצרת breaklist מסוג רשימת התחייבות. מבני בקרה המאפשרים break יטפלו ב- breaklist של בניהם עפ"י משמעות המבנה (בד"כ יוכנסו ל- nextlist שלהם). מבני בקרה פשוטים יאחדו את רשימות ה- breaklist של בניהם לרשימת ה- breaklist שלהם.

נשנה קצת את הדקדוק (כי עכשיו break הוא פקודה):

$S \rightarrow$ <u>switch</u> E <u>begin</u> L D <u>end</u>
$L \rightarrow L\ C \mid C$

C → case V : S ;
D → default : S ; | ε
V → const
S → break

גם למשתנים C, D ו- L נוסיף את התכונה breaklist.
נניח כי כאשר התכונה breaklist לא מאותחלת מפורשות, היא מאותחלת לרשימה ריקה.

סכמת התרגום:

(1) S → switch E begin G L D end

```
{ backpatch(G.nextlist, nextquad() );
  for (k=0; k < L.val_num; k++)
    emit('if' E.place '=' pop(L.vals) 'goto'
          pop(L.quads));
  if (D.with_default == true) {
    emit('goto' D.quad);
    backpatch(L.nextlist, D.quad);
    S.nextlist =
      merge(L.breaklist, D.nextlist, D.breaklist);
  }
  else {
     S.nextlist = merge(L.nextlist, L.breaklist);
  }
}
```

(2) G → ε

```
{ G.nextlist = makelist(nextquad());
  emit('goto ___');
}
```

(3) L → $L_1$ C

```
{ backpatch(L1.nextlist, C.quad);
  L.nextlist = C.nextlist;
  L.breaklist = merge(L1.breaklist, C.breaklist);
  L.val_num = L1.val_num + 1;
  L.vals = L1.vals;
  push(L.vals, C.val);
  L.quads = L1.quads;
  push(L.quads, C.quad);
}
```

(4) L → C

```
{ L.nextlist = C.nextlist;
  L.breaklist = C.breaklist;
  L.val_num = 1;
  L.vals = newstack();
  push(L.vals, C.val);
  L.quads = newstack();
  push(L.quads, C.quad);
}
```

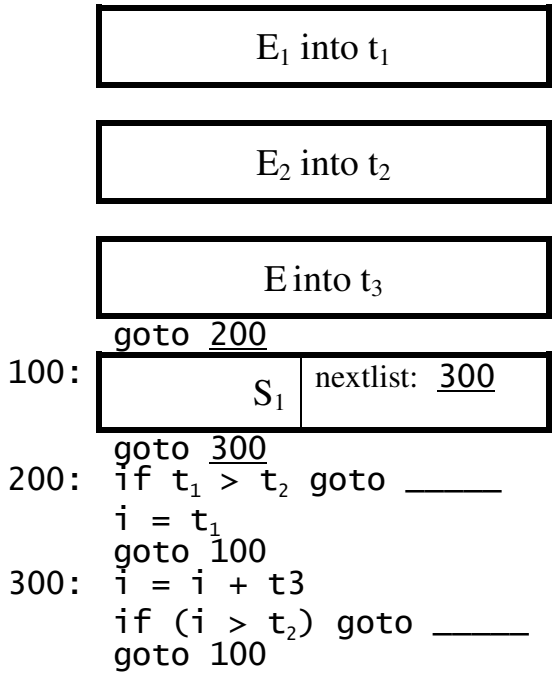
(5) C → case V : Q S ;

```
{ C.nextlist = S.nextlist;
  C.breaklist = S.breaklist;
  C.quad = Q.quad;
  C.val = V.val;
}
```

| | | |
|---|---|---|
| (6) | D → default : Q S ; | { D.nextlist = merge(S.nextlist, makelist(nextquad()));<br>emit('goto ___');<br>D.breaklist = S.breaklist;<br>D.quad = N.quad;<br>D.with_default = true;<br>} |
| (7) | Q → ε | { Q.quad = nextquad(); } |
| (8) | D → ε | { D.with_default = false; } |
| (9) | V → const | { V.val = const.val; } |
| (10) | S → break | { S.breaklist = makelist(nextquad());<br>emit('goto ___');<br>} |

**שאלה 2:**

א. עבור מבנה הבקרה המוגדר בתרגיל יכולות להיות פריסות קוד שונות בגלל שכבר בזמן קומפילציה ידוע האם צריך לבצע את החישוב של STEP או להשתמש בערך ברירת המחדל שלו וגם האם צריך לקדם או להפחית את ערכו של id בכל איטרציה.
כאן נביא פריסת קוד עבור המקרה שערכו של id מקודם.
נניח כי ערך התכונה id.place הוא i.




במקרה ש- STEP → ε ירשם כאן $t_3 = 1$.

ב- downto התנאי יהיה $t_1 < t_2$

ב- downto יהיה - (במקום +)

## ב. סכמת התרגום:

```
S     ->    for id assign E1 TO E2 STEP G1 Q S1 G2
                {
                        backpatch(G1.nextlist, nextquad());
                        list tmplist = makelist(nextquad());
                        if (TO.up)
                                emit('if' E1.place '>' E2.place 'goto ____');
                        else
                                emit('if' E1.place '<' E2.place 'goto ____');
                        emit(id.place '=' E1.place);
                        emit('goto' Q.quad);
                        backpatch(S1.nextlist, nextquad());
                        backpatch(G2.nextlist, nextquad());
                        if (TO.up)
                                emit(id.place '=' id.place '+' STEP.place);
                        else
                                emit(id.place '=' id.place '-' STEP.place);
                        S.nextlist = merge(tmplist, makelist(nextquad()));
                        if (TO.up)
                                emit('if' id.place '>' E2.place 'goto ____');
                        else
                                emit('if' id.place '<' E2.place 'goto ____');
                        emit('goto' Q.quad);
                }

TO    ->    to      { TO.up = true; }

TO    ->    downto          { TO.up = false; }

STEP  ->    step E  { STEP.place = E.place; }

STEP  ->    ε       { STEP.place = newtemp();
                      emit(STEP.place '= 1');
                }

G     ->    ε       { G.nextlist = makelist(nextquad());
                      emit('goto ____');
                }

Q     ->    ε       { Q.quad = nextquad(); }
```

ג. כאשר ערכם של $E_1$, $E_2$ ו- STEP ידועים בזמן קומפילציה, אפשר לדעת בזמן קומפילציה את מספר האיטרציות שיבוצעו אם ידוע כיצד $S_1$ משפיע על id.
(בפרט אם הקומפיילר מזהה כי $S_1$ לא משפיע על id, אז מספר האיטרציות ידוע).