שיטות הידור 046266

# פרויקט – חלק ג'

מגישים: עידן גולן 204626444
דולב שמע 204559744

## מימוש הקומפיילר

- לצורך מימוש הקומפיילר בחלק השלישי והאחרון של הפרויקט, נעזרנו במימושים של החלקים הקודמים: ניתוח לקסיקלי (יצירת token-ים) באמצעות flex, ניתוח תחבירי (יצירת ה-symbols וה-parser tree) באמצעות Bison.
- כדי לתמוך בתכונות שונות עבור צמתים שונים בעץ הפריסה, יצרנו מחלקות שונות (שיורשות ממחלקה וירטואלית משותפת) לכל סט צמתים בעלי תכונות משותפות.
- הוספנו ניתוח סמנטי באמצעות Bison, וכן יצרנו ותחזקנו טבלאות סמלים שונות הנדרשות לקבלת ההקשר (כגון טיפוסי משתנים, קיומן ומיקומן של פונקציות וכד') הנדרש ליצירת קוד הסף.
- את הקוד הזרקנו לתוך מערך דינמי (std::vector) בו כל איבר הוא שורת קוד, כך שע"י ביצוע פעולת emit אנו מוסיפים שורות קוד בסדר כרונולוגי ובכל רגע נתון ניתן לקרוא את מספר שורת הקוד הבא ע"י פעולת nextQuad. כמו כן, אנו מבצעים הטלאה לאחור (backpatching) לשורות בהן כתובת הקפיצה טרם הייתה ידועה בזמן הכנסתן, ע"י גישה ישירה למערך זה (על בסיס רשימת שורות קוד הדורשות הטלאה, העוברות כתכונה נוצרת בתהליך הגזירה של Bison) ושכתוב כתובת ה-placeholder שהוטמעה בפקודת ה-jump.

## אופן הקצאת רגיסטרים מיוחדים

- RA – כתובת הקפיצה בחזרה מפונקציה (I0)
- FP – מצביע לתחילת רשומת ההפעלה (I1, F0)
- SP – מצביע לכתובת הבאה הפנויה לכתיבה במחסנית (I2, F1)
- RT – רגיסטר לשמירת ערך חזרה מפונקציה (I3, F2)
- DP – מצביע למשתנה הראשון במחסנית ברשומת ההפעלה הנוכחית. (declaration pointer) (I4, F3)

נעשה שימוש ברגיסטרים של INT וFLOAT לכל תפקיד (למעט RA) כדי לתמוך בפקודות ASM על שני סוגי הרגיסטרים. מכיוון שמהות ערכי הרגיסטרים הנ"ל היא כתובת במחסנית, ערכם מתוחזק ברגיסטרי הINT בלבד; כאשר נדרשת קריאת ערכים אלה בפעולת FLOAT, מבוצע cast מרגיסטר ה-INT המתאים לרגיסטר ה-FLOAT הרלוונטי.

יתר הרגיסטרים משמשים משתנים זמניים.

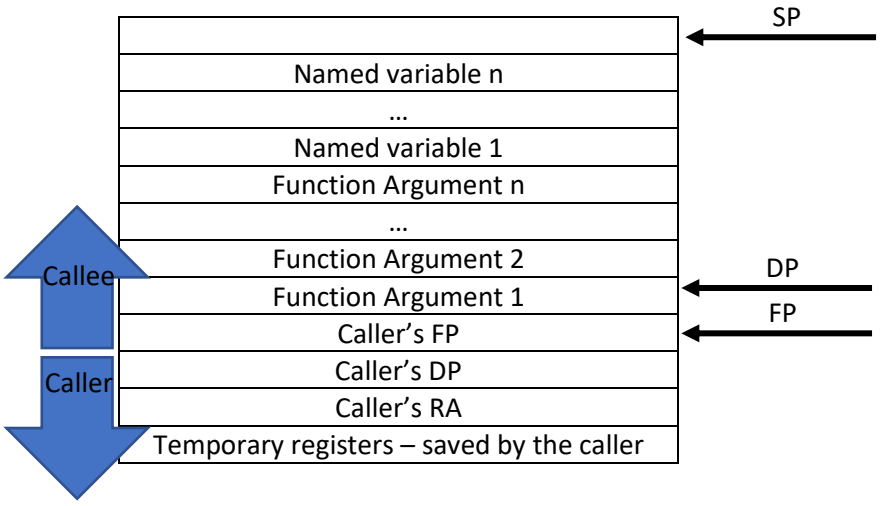
## מבני הנתונים בשימוש

- באופן כללי, מימשנו את מבני הנתונים השונים באמצעות מבני LIST, MAP, VECTOR מתוך C++ STL.
- <u>צמתי עץ הפריסה</u> – טיפוס הצמתים (המוגדר ב-Bison ע"י YYSTYPE) הוא מצביע למחלקה וירטואלית ParserNode. ממחלקה זו יורשות 2 מחלקות "בנים": NodeToken ו-NodeSymbol המשמשות את הצמתים הנוצרים מאסימונים וסימבולים, בהתאמה. כמו כן, מ-NodeSymbol נורשות מחלקות נוספות לסימבולים השונים לפי תכונות נדרשות: nextlist, breaklist, place, expression type, ועוד.
- <u>טבלת פונקציות גלובלית</u> – מכילה:

- טבלה (std::map) הממפה שם של פונקציה (string) לאובייקט מסוג funcEntry, המכיל את מאפייני הפונקציה: שם הפונקציה, טיפוס ההחזרה שלה, רשימת שמות וטיפוסי הפרמטרים שמקבלת, שורת הגדרתה בקוד והאם הוגדרה.
- מצביע לאובייקט ה-funcEntry שתואר לעיל, השייך לפונקציה המוגדרת בקטע הקוד הנוכחי. כך ניתן לבצע אכיפה סמנטית, למשל על טיפוס ערך ההחזרה בגזירת הוראת return.

- <u>טבלת משתנים גלובלית</u> – מכילה:
  - רשימת זוגות (עבור טיפוסי int ו-float) של טבלאות משתנים לכל תחום (scope), כך שבראש הרשימה נמצא התחום הנוכחי (הפנימי ביותר) ובסופה תחום הפונקציה.
    - <u>בכל טבלת משתנים</u> (עבור תחום), מנוהלות ההקצאות של המשתנים הזמניים והמשתנים שהוכרזו (named variables):
      - <u>משתנים זמניים</u>: מכיוון שבכל register file הרגיסטרים השמורים הם באינדקסים הנמוכים, אנו מחזיקים מצביע לרגיסטר הפנוי הבא. בעת בקשת משתנה זמני חדש, מצביע זה יועבר למבקש ויקודם למקום הפנוי הבא.
      - <u>משתנים מוכרזים</u>: ממפים (std::map) שמות משתנים למספר המייצג את ההיסט במחסנית בו הוקצה משתנה זה, יחסית לערך השמור ברגיסטר DP שתואר לעיל.
        - כאשר נדרשת גישה למשתנה מוכרז לפי שם, החיפוש מתבצע בצורה איטרטיבית על רשימת טבלאות המשתנים, החל מהתחום הפנימי ביותר ועד לחיצוני, כך שיוחזרו פרטיו של המשתנה הפנימי ביותר.
        - בעת הכרזה על משתנה חדש, מבוצע חיפוש עבור הכרזה על משתנה עם שם זהה בתחום הנוכחי (הפנימי) בלבד, כדי למנוע הכרזות כפולות.

## מבנה רשומת הפעלה

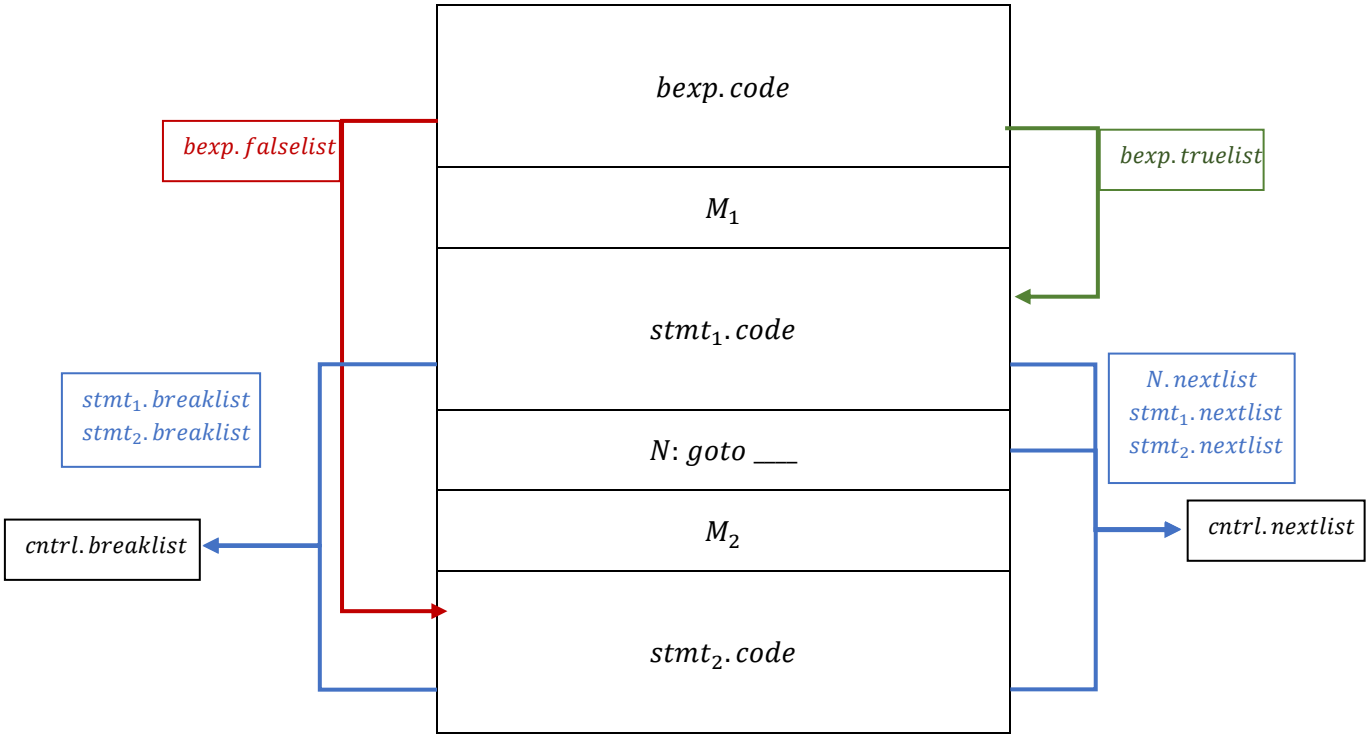



- בכל הכרזה על משתנה חדש, המחסנית גדלה ע"י הגדלת SP.
- בקריאה לפונקציה, הפונקציה הקוראת מאחסנת במחסנית את ערכי הרגיסטרים השמורים שלה, את המשתנים הזמניים והארגומנטים המועברים לפונקציה הנקראת.
- בחזרה מפונקציה, הפונקציה הקוראת משחזרת את ערכי הרגיסטרים השמורים והמשתנים הזמניים.

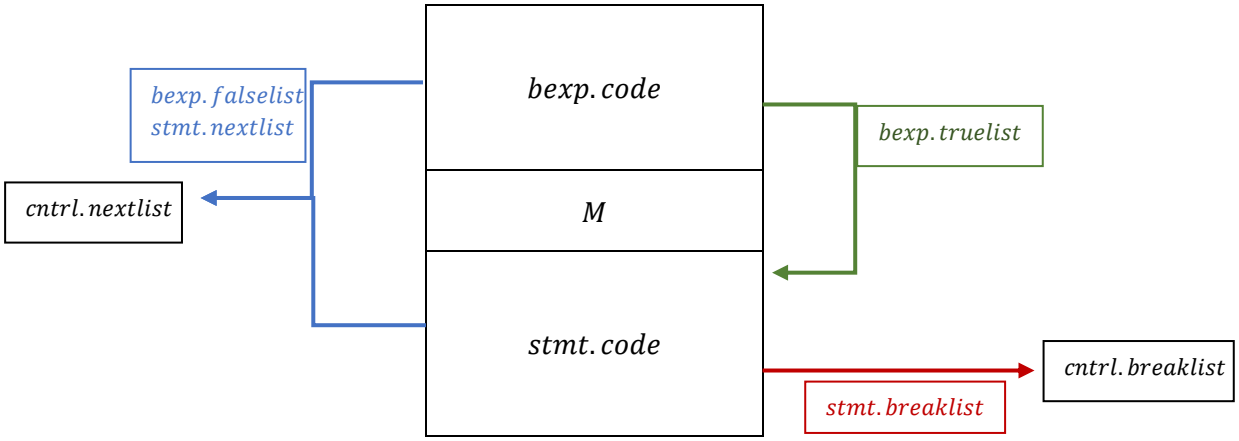
## הטלאה לאחור (backpatching)

להלן תיאור השימוש בbackpatching בכללים הסמנטיים במבני הבקרה השונים:
(סימבולים מסומנים באותיות קטנות ומרקרים ואסימונים באותיות גדולות)

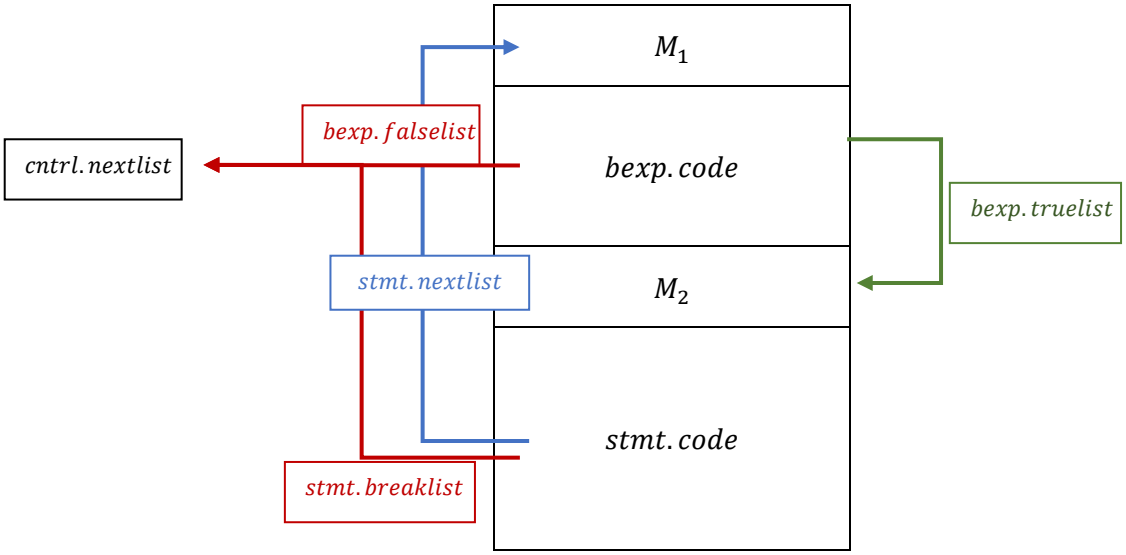
- **cntrl -> IF bexp THEN M1 stmt ELSE N M2 stmt**



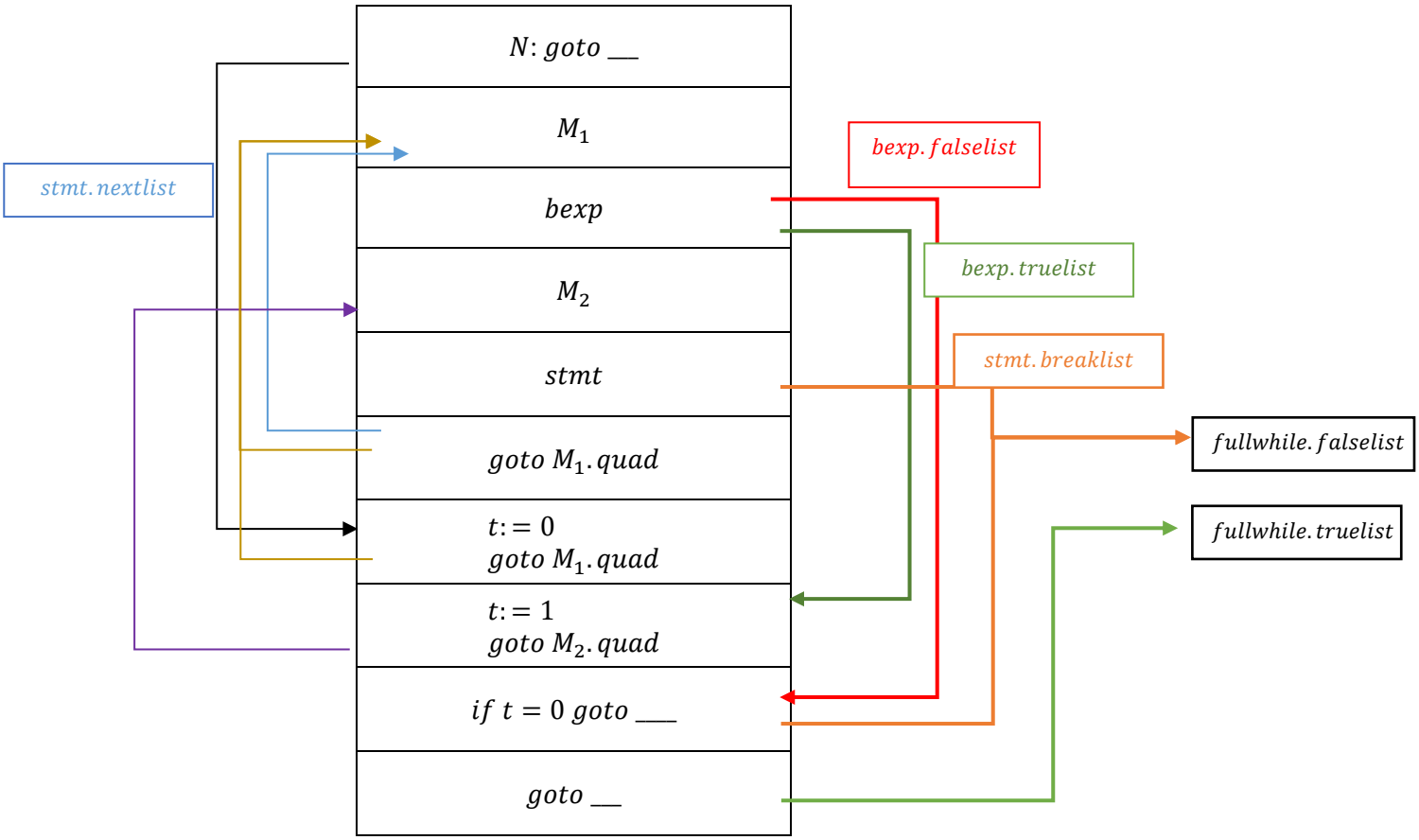

- **cntrl -> IF bexp THEN M stmt**



- **cntrl -> WHILE M bexp DO M stmt**

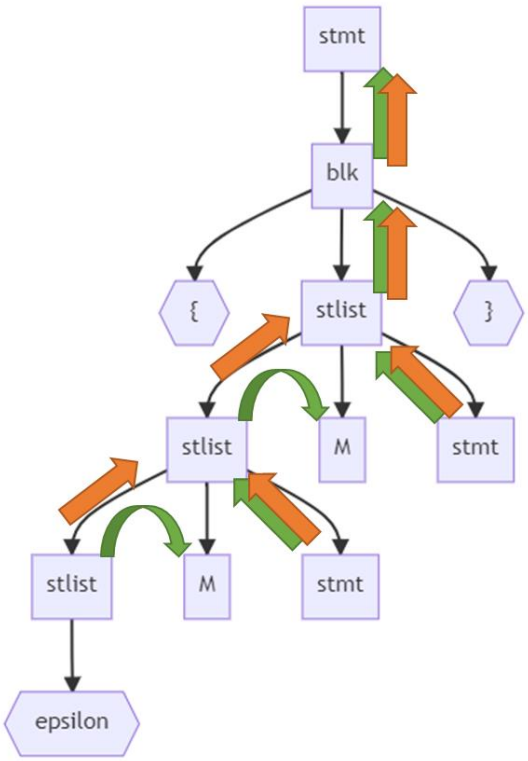



- **fullwhile -> FULL_WHILE N M1 bexp DO M2 stmt**




**הערה**: בגזירת cntrl->fullwhile רשימות ה-truelist ו-falselist ממוזגות לתוך רשימת ה-nextlist של cntrl; בגזירת bexp->fullwhile רשימות ה-truelist ו-falselist מועברות כפי שהן לרשימות truelist ו-falselist של bexp בהתאמה.

האיור הבא מתאר את מעבר התכונות הנוצרות breaklist ו-nextlist של צמתים מסוג statement על גבי עץ פריסה לדוגמה. חץ כתום מסמל את העברת התכונה הנוצרת breaklist, חץ ירוק את התכונה nextlist, וחץ ירוק מעוגל את ביצוע ה-backpatch של nextlist.




לא נעשה שימוש במבני נתונים חיצוניים כלשהם (למעט כאמור, ב-STL).