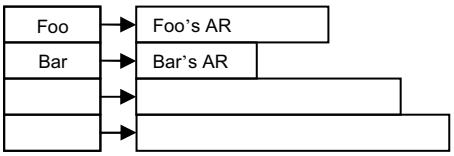
תאריך: 25.2.2009
מועד: א'
מרצה: ד"ר רן שחם
מתרגל: אוהד שחם
חומר: פתוח
משך: 3 שעות

מבחן בקומפילציה

שאלה 1 (20 נקודות) : רשומות הפעלה

אנו מעוניינים כי רשומות ההפעלה ינוהלו במערך ולא במחסנית כמתואר בציור הבא. במערך זה קיימת כניסה יחידה לכל פונקציה בתכנית. בכל כניסה במערך ניתן לנהל רשומת הפעלה אחת בלבד. שים לב שגודלה של רשומת הפעלה עלול להשתנות במהלך ריצת התוכנית. הנח כי כל קוד התכנית זמין בקובץ יחיד.



א. (12 נק') הצע דרך למימוש משטר מערך מעין זה. לאילו מבני נתונים נוספים תזדקק. התייחס לכל רכיבי רשומת ההפעלה – מה ישתנה במימוש עבורם ומה יוותר ללא שינוי בהשוואה למימוש משטר מחסנית.
ב. (3 נק') לאילו שינויים תזדקק אם קוד התכנית מחולק למספר קבצים.
ג. (5 נק') האם ניתן להריץ כל תכנית במשטר מערך. הבחן בין 3 מקרים אפשריים.

פתרון:
א. להלן הצעה למימוש:
המימוש יהיה כמו שיטוח של ה-stack במימוש הרגיל. נשתמש במבנה נתונים חדש המחזיק את כל נתוני ה-frame כפי שיפורט.
- המתודה הקוראת מבצעת את הפעולות הבאות:
1. שומרת את כל ה-caller saved-registers בסוף ה-frame שלה
2. שומרת את ה-frame של ה-callee ב-heap (במידה וקיים – כאשר ה-callee נקראה קודם), ושומרת את הכתובת ב-heap בסוף ה-frame שלה. הגישה ל-callee הראשונית היא פשוט גישה למקום במערך המתאים ל-callee.
3. מאתחלת frame חדש ל-callee ודוחפת לתחילתו את:
   a. כל הפרמטרים (ניתן בסדר הפוך כדי לשמור כמה שיותר דומה לקודם)
   b. Return address (הפעולה הבאה שיש לבצע בסוף המתודה)
   c. ה-FP (של ה-caller)
   d. ה-SP (של ה-caller, כרגע מצביע למקום המחזיק את ה-address ב-heap של ה-callee הקודמת)
4. עדכון ה-SP וה-FP להיות סוף ה-frame של ה-callee (טרם הוכנסו לוקאליים וכו').
- המתודה הנקראת מבצעת את הפעולות הבאות:
1. שומרת את כל ה-locals
2. שומרת את כל ה-callee saved registers
3. מבצעת את הפעולות שצריכה

שחזור:
4. משחזרת את הרגיסטרים שהשתמשה בהם ומשחררת את הזיכרון שתפסו
5. משחררת את הזיכרון של ה-locals שהשתמשה בו
6. משחזרת את ה-FP
7. משחזרת את ה-return address לתוך רגיסטר TMP
8. משחזרת את ה-SP (כעת אין מצביעים ל-frame של ה-callee פרט ל-TMP שמצביע ל-return address)
9. קפיצה ל-return address מה-TMP
- המתודה הקוראת:
1. משחררת את הזיכרון של ה-callee
2. משחזרת מה-heap את ה-frame של ה-callee הקודמת
3. משחזרת את ה-caller saved registers

ב. ה-linker יצטרך לשרשר יחד את כל מערכי ה-frames של כל הקבצים.

ג. אם לא היינו דורשים שמירה ב-heap של frame של ה-callee לפני שימוש בו, לא היינו יכולים להריץ קוד עם קריאות רקורסיביות למתודות. מקרים נוספים בעייתיים: ?

שאלה 2 (25 נקודות) : ניתוח תחבירי

נתון הדקדוק הבא:

E -> *E | &E | E=E | E→E | id

א. (2 נק') הראי כי הדקדוק הוא רב משמעי.
ב. (5 נק') בני דקדוק חד משמעי המקבל את אותה שפה.
ג. (5 נק') בני אוטומט עבור הדקדוק המקורי.
ד. (5 נק') בני parsing table מתוך האוטומט בצורה שתייצג גם shift / reduce conflict.
ה. (5 נק') הריצי את ה parser שבנית בסעיף ד' על הקלט " id = *&id→&*id" תוך מתן עדיפות ל reduce על פני shift.
ו. (3 נק') האם ב parser שהתקבל בסעיף ד' האופרטור → בעל עדיפות על פני שאר האופרטורים בדקדוק?

פתרון:

א. ניתן לבנות בקלות שני עצים שאחד נוטה ימינה ושני שמאלה עבור הקלט "id=id=id".

ב. להלן דקדוק חד משמעי:

$E \rightarrow * E \mid \&E \mid E = T \mid T$
$T \rightarrow \ T \rightarrow F \mid F$
$F \rightarrow id$

ג. נו, בונים אוטומט.

ד. להלן parsing table:

State	$	*	&	=	→	id	E
0	Err	S2	S3	Err	Err	S4	S1
1	S5	Err	Err	S6	S7	Err	Err
2	Err	S2	S3	Err	Err	S4	S8
3	Err	S2	S3	Err	Err	S4	S9
4	r$E \rightarrow id$						
5	r$S \rightarrow E\$$						
6	Err	S2	S3	Err	Err	S4	S10
7	Err	S2	S3	Err	Err	S4	S11
8	r$E \rightarrow * E$	r$E \rightarrow * E$	r$E \rightarrow * E$	r$E \rightarrow * E$ S6	r$E \rightarrow * E$ S7	r$E \rightarrow * E$	r$E \rightarrow * E$
9	r$E \rightarrow \&E$	r$E \rightarrow \&E$	r$E \rightarrow \&E$	r$E \rightarrow \&E$ S6	r$E \rightarrow \&E$ S7	r$E \rightarrow \&E$	r$E \rightarrow \&E$
10	r$E \rightarrow E = E$	r$E \rightarrow E = E$	r$E \rightarrow E = E$	$rE \rightarrow E = E$ S6	r$E \rightarrow E = E$ S7	$rE \rightarrow E = E$	$rE \rightarrow E = E$
11	r$E \rightarrow E \rightarrow E$	r$E \rightarrow E \rightarrow E$	r$E \rightarrow E \rightarrow E$	r$E \rightarrow E \rightarrow E$ S6	r$E \rightarrow E \rightarrow E$ S7	r$E \rightarrow E \rightarrow E$	r$E \rightarrow E \rightarrow E$

קונפליקטים shift-reduce מסומנים בטבלה לעיל.

ה. להלן ריצה לקלט: "$id = \$\&id \rightarrow \& * id$":

Stack (fills left to right)	Input	Action
$0(\$)$	$id = * \&id \rightarrow \& * id\$$	Shift 4
$0(\$), 4(id)$	$= * \&id \rightarrow \& * id\$$	Reduce $E \rightarrow id$
$0(\$), 1(E)$	$= * \&id \rightarrow \& * id\$$	Shift 6
$0(\$), 1(E), 6(=)$	$* \&id \rightarrow \& * id\$$	Shift 2
$0(\$), 1(E), 6(=), 2(*)$	$\&id \rightarrow \& * id\$$	Shift 3
$0(\$), 1(E), 6(=), 2(*), 3(\&)$	$id \rightarrow \& * id\$$	Shift 4
$0(\$), 1(E), 6(=), 2(*), 3(\&), 4(id)$	$\rightarrow \& * id\$$	reduce $E \rightarrow id$
$0(\$), 1(E), 6(=), 2(*), 3(\&), 9(E)$	$\rightarrow \& * id\$$	Reduce $E \rightarrow \&E$
$0(\$), 1(E), 6(=), 2(*), 8(E)$	$\rightarrow \& * id\$$	Shift 7
$0(\$), 1(E), 6(=), 2(*), 8(E), 7(\rightarrow)$	$\& * id\$$	Shift 3
$0(\$), 1(E), 6(=), 2(*), 8(E), 7(\rightarrow), 3(\&)$	$* id\$$	Shift 2
$0(\$), 1(E), 6(=), 2(*), 8(E), 7(\rightarrow), 3(\&), 2(*)$	$id\$$	Shift 4
$0(\$), 1(E), 6(=), 2(*), 8(E), 7(\rightarrow), 3(\&), 2(*), 4(id)$	$\$$	Reduce $E \rightarrow id$
$0(\$), 1(E), 6(=), 2(*), 8(E), 7(\rightarrow), 3(\&), 2(*), 8(E)$	$\$$	Reduce $E \rightarrow * E$
$0(\$), 1(E), 6(=), 2(*), 8(E), 7(\rightarrow), 3(\&), 9(E)$	$\$$	Reduce $E \rightarrow \&E$
$0(\$), 1(E), 6(=), 2(*), 8(E), 7(\rightarrow), 11(E)$	$\$$	Reduce $E \rightarrow E \rightarrow E$
$0(\$), 1(E), 6(=), 2(*), 8(E)$	$\$$	Reduce $E \rightarrow * E$
$0(\$), 1(E), 6(=), 10(E)$	$\$$	Reduce $E \rightarrow E = E$
$0(\$), 1(E)$	$\$$	Shift 5
$0(\$), 1(E), 5(\$)$		Reduce $S \rightarrow E\$$
$0(\$)$		Accept

ו. לאופרטור → לא תהיה עדיפות, להיפך – לאופרטורים אונאריים תהיה עדיפות על פני הבינאריים, בין היתר עליו. למשל עבור הקלט $id \rightarrow * id$ לא תהיה גזירה $E \rightarrow E \rightarrow E$ עד אשר יהיו הגזירות $E \rightarrow * E$ (לשני הצדדים).

שאלה 3 (35 נקודות) : הרחבה לשפה

שפת IC איננה תומכת ב function overloading ולכן לא חוקי ב IC לכתוב באותו scope שתי מתודות בעלות אותו שם שמקבלות רשימת פרמטרים מועברים שונה כמו בדוגמא הבאה:

```
Class A {                          class C {
       int     foo(C c1, C c2);            …
       boolean foo(C c, D d);
}
                                   class D {
                                          …
                                   }
```

א. (10 נק') כתבי אלגוריתם שמוסיף תמיכה ב function overloading בזמן קומפילציה ללא תמיכת זמן ריצה. והסבירי באיזה שלב צריך לבצע את האלגוריתם ואיזה מבני נתונים צריכים להשתנות.

ב. (5 נק') האם האלגוריתם שסיפקת בסעיף א' יתמוך במקרה ש class D יורש מ class C ? הראי דוגמא לתוכנית שתעורר את הבעיה.

ג. (10 נק') הראי אלגוריתם שפותר את הבעיה מהסעיף הקודם והסבירי באיזה שלב צריך לבצע את האלגוריתם ואילו מבני נתונים צריכים להשתנות.

ד. (10 נק') הסבירי מה החסרונות של האלגוריתם מסעיף ג' האם ניתן למזג את האלגוריתם מסעיף א' ומסעיף ג' בשביל לשפר את האלגוריתם מסעיף ג'? אם כן, הראי כיצד והסבירי אילו מבני נתונים צריכים להשתנות.

פתרון:

א. השינוי צריך להיעשות עד שלב ה-LIR (לא כולל) באופן הבא:
- בשלב ה-Lexical Analysis: אין שינויים כיוון שאין tokens חדשים בשפה.
- בשלב ה-Syntax Analysis: גם כאן אין שינויים.
- בשלב ה-Semantic Analysis / checks:
נוסיף ל-global symbol table מבנה נתונים נוסף שיחזיק לכל מתודה hash table שה-key שלה הוא method type (מה-type table) והערך שלה הוא שם המתודה משורשר למספר עולה סדרתית. למשל, עבור foo יהיו כניסות:

Int->int	0_foo
Int->bool	1_foo

נשים לב שלא יתכן קלט מהמשתמש בשם זה למתודה כי זה לא חוקי בזמן ה-parsing. מבנה זה יהיה מאותחל להיות ריק. בזמן בניית ה-symbol table:

- כאשר נתקלים ב-method, בודקים האם יש לה כניסה במבנה הנתונים הזה. אם לא, ניצור כניסה חדשה עם hash table חדשה, וניצור כניסה עם ה-type והשם החדש, נניח 0_foo. **נשנה** את ה-method name ב-ASTNode להיות השם החדש 0_foo.
נציין כי בשלב זה, כאשר בודקים חוקיות של הגדרת המתודה, נצטרך לבדוק רק שלא קיימת באותה מחלקה מתודה עם חתימה זהה לחלוטין (מבחינת שם ו-type), בניגוד לבדיקה הקודמת האוסרת overloading.
אם נתקלנו במתודה שיש לה כבר כניסה במבנה הנתונים החדש, אם יש ב-hash table כניסה עם ה-type, נשנה את שמה לשם החדש. אם לא, נכניס רשומה חדשה ל-hash table עם ה-type החדש ושם חדש: 1_foo וכן הלאה.
- כאשר נתקלים ב-VirtualCall ,StaticCall (ניתן גם ל-LibraryCall), בבדיקת ה-type, נבדוק תחילה במבנה הנתונים החדש אם ה-type מופיע עבור שם המתודה. אם כן, **נשנה** את שם המתודה ב-call לשם המתודה החדש, ונמשיך בבדיקה.

קוד ה-LIR יגונרט בדיוק באותו אופן כמו קודם, שכן כעת חזרנו למצב הקודם ולכל היותר יהיו מתודות overriding (החלפת השמות "הסירה" Overloading).

ב. כן, אנו תומכים.

ג. בדיוק האלגוריתם ב-א'.

ד. אין בעיה עם האלגוריתם המוצע כיוון שהוא מראש מטפל במקרי ירושה ולא מתקן אלגוריתם שלא תומך.

שאלה 4 (20 נק') : הקצאת אוגרים גלובלית

היזכר באלגוריתם הקצאת האוגרים הגלובאלי ע"י צביעת גרף ההפרעות. נמק את הפתרונות בצורה מדויקת ככל האפשר, ע"י התייחסות לאלגוריתם.

א. (10 נק') אנו מעוניינים למצוא חסמים על ה-k המינימלי עבורו גרף ההפרעות הוא k-צביע. נסמן ערך זה ב-k_{min}.
הצע דרך למציאת חסמים a,b בזמן פולינומיאלי. היעזר ב liveness information וגרף ההפרעות על מנת להציע החסמים.
הערה: המנע מהצעת חסמים טריביאליים כגון 0 כחסם תחתון או מספר צמתי הגרף כחסם עליון. כמו כן, לא ניתן להפעיל את אלגוריתם הצביעה על מנת להציע החסמים.

ב. (10 נק') כזכור, Conservative Coalescing הינה היוריסטיקה לחיסכון בפקודות move אשר מופעלת בין שלב simplify לשלב ה-potential spill. הצע היוריסטיקה חלופית לחסכון בפקודות move אשר מופעלת בשלב ה-select.
הבחן בין שני מקרים אפשריים ותן כלל לכל אחד מהם.

פתרון:

א.

חסם תחתון: נבחר את השורה בקוד שיש בה את הכי הרבה משתנים ורגיסטרים סימבוליים חיים – מספר זה יהיה חסם תחתון. נשים לב כי יהיו מקרים בהם זה לא טוב (אם a,b חיים בשורה אחת נקבל 2, אבל אם שתי שורות אחרי יש a,c ו-b,c נצטרך 3).
חסם עליון: נבחר את דרגת הצומת עם הדרגה המקסימלית + 1: הקליקה הגדולה ביותר תהיה בגודל זה, ואם כיסינו את הקליקה הגדולה ביותר, כיסינו את כל הגרף.

ב.

נציע את האלגוריתם הבא:
נבצע את השלבים של ה-potential spill וה-simplify כמו קודם רק ללא coalescing. ואז בשלב ה-select:

- אם נוצר actual spill מתחילים מחדש (כי הוא לא יכול להיכנס לאף רגיסטר).
- אחרת: אם לצומת שהוצאנו מהמחסנית אין קשתות מקווקוות, נבחר רגיסטר שאין לו קשת רציפה אליו ונדחוף אותו אליו.
אם יש לו קשתות מקווקוות, תבחר רגיסטר שיש לו קשת מקווקוות אליו (ולא רציפה) ונדחוף אותו אליו.
במידה ויש כמה רגיסטרים שמחוברים אליו עם קשתות מקווקוות, נבחר את הרגיסטר שיש אליו את המספר המקסימלי של קשתות מקווקוות.
הערה: יתכן מצב של כמה קשתות מקווקוות בין משתנה לרגיסטר, במידה ומשתנה היה מקושר בקיווקוו למשתנה אחר ולרגיסטר, והמשתנה האחר הוכנס לרגיסטר – למשתנה הראשון יהיו שתי קשתות מקווקוות לרגיסטר.

תאריך: 14.9.2009
מועד: ב'
מרצה: ד"ר רן שחם
מתרגל: אוהד שחם
חומר: פתוח
משך: 3 שעות

מבחן בקומפילציה

שאלה 1 (25 נקודות) : סיווג מאורעות

נתונה רשימת מאורעות. לכל אחד מהמאורעות צייני בקצרה (מספיקות 3-2 שורות הסבר לכל פריט):
1) האם הוא מתרחש בזמן ריצה, בזמן קומפילציה, או בזמן בניית הקומפיילר;
2) אם בחרת בזמן קומפילציה, מהו השלב המסוים בקומפיילר בו מתרחש המאורע;
3) מהם מבני הנתונים והאלגוריתמים הרלבנטיים.
במידה ויש מספר תשובות נכונות, הסבירי את כולן.

א. (5 נק') הפקודה break נמצאת בתוך לולאת while
ב. (5 נק') מתקבלת הודעת שגיאה שהדקדוק הינו רב משמעי
ג. (5 נק') ה offset של הפונקציה foo ב dispatch vector של מחלקה C הוא 3
ד. (5 נק') ה tokens החוקיים בשפה הם: … while, if, for
ה. (5 נק') הקריאה c.bar(3) איננה חוקית ולכן מופיעה הודעת שגיאה

פתרון:
א. בשלב קומפילציה: בשלב הבדיקות הסמנטיות. מבנה הנתונים הרלוונטי הוא ה-AST והאלגוריתם בודק בהתקלות ב-break שהוא מוכל בתוך while באמצעות visitor.
ב. בשלב בניית הקומפיילר: משתמש ברשימת החוקים של הדקדוק (יתכן שהמימוש הפנימי משתמש בטבלת הגזירה).
ג. בשלב קומפילציה: נעשה בשלב התרגום ל-LIR, משתמשים במבנה ה-ClassLayout כדי למצוא את ה-offset.
ד. שלב בניית הקומפיילר: משתמשים ב-lexer כדי לפרק את הקלט ל-tokens.
ה. שתי אפשרויות:
- בשלב קומפילציה: בשלב ה-type check תתכן שגיאה, למשל אם c.bar לא מקבלת int. משתמשים ב-type table כדי למצוא את ה-types וב-symbol table כדי להוציא את נתוני ה-type על כל חלקי הביטוי.
- בזמן ריצה: אם למשל c היא null נקבל null pointer exception.

שאלה 2 (25 נקודות) : ניתוח תחבירי

נתון הדקדוק הבא:
S -> E$
E -> id | id (E)

א. (5 נק') הראי כי הדקדוק אינו LR(0).
ב. (5 נק') בני דקדוק LR(0) המקבל את אותה שפה. **רמז:** כתבי כלל מיוחד עבור המחרוזת "id$"
ג. (5 נק') בני אוטומט עבור הדקדוק של סעיף ב'.
ד. (5 נק') בני parsing table מתוך האוטומט.
ה. (5 נק') הריצי את ה parser שבנית בסעיף ד' על הקלט "id(id(id))$"

פתרון:
נו...

שאלה 3 (35 נקודות) : הרחבה לשפה

המילה השמורה super ב Java משמשת לקריאה ל constructor של ה super class וחייבת להופיע כפקודה ראשונה ב constructor. בדוגמא הבאה, ה constructor של A נקרא בתחילת ה constructor של B.

א. (5 נק') הסבירי מהם השינויים הנדרשים בכדי להוסיף לשפת IC תמיכה ב constructors.

ב. (15 נק') הוסיפי לשפת IC תמיכה בקריאה ל constructor של ה super class כאשר הפקודה ()super מופיעה כפקודה ראשונה ב constructor. הניחי שה constructors הינם ללא פרמטרים.

```
Class A {
        A() {
                …
        }

        void foo() {
                …
        }
}
```

```
class B extends A {
        B() {
                super();
                …
        }

        void foo() {
                …
        }

        void bar() {
                super.foo();
                …
        }
}
```

בנוסף לשימוש ב super לקריאת ה constructor של ה super class. ניתן להשתמש ב super בכדי לקרוא לפונקציות override מה super class. בדוגמא, פונקצית foo מ class A נקראת על ידי פונקצית bar מ class B על ידי שימוש בפקודה super.foo().

ג. (15 נק') הוסף לשפת IC תמיכה ב super על מנת לקרוא לפונקציות מה super class.

פתרון:

א. השינויים הנדרשים בקוד:

שלב ה-lexer: אין שינויים.

שלב ה-parser: הוספת כלל גזירה עבור constructor ועבור יצירת instance של האובייקט (במידה ומאפשרים קבלת פרמטרים בקונסטרקציה). נתייחס ב-AST ל-constructor כמתודה שערך ההחזרה שלה הוא null.

שלב ה-semantics:

- בדיקה לכל מתודה אם ערך ההחזרה הוא null – אם כן לבדוק ששם המתודה הוא שם ה-class.
- בדיקה בכל מופע של new CLASS_ID שמתקבל מספר הפרמטרים הנדרש ומהסוג הנכון.

שלב התרגום ל-LIR:

- בתרגום פקודת new, מוסיפים קוד כמו שרצים במקרה של קריאה וירטואלית עם מתודת הבנאי (ניתן לעשות זאת ע"י קריאת accept על צומת חדש מסוג NewObject עם הפרמטרים הנדרשים).

שלב התרגום לשפת מכונה:

השינויים ב-LIR גוררים את השינויים הנדרשים ב-assembly.

ב.

שלב ה-lexer: נוסיף token מסוג SUPER

שלב ה-parser: נוסיף את כלל הגזירה הבא: CLASS_ID:name LP RP LCBR stmt_list RCBR או CLASS_ID:name LP RP LCBR SUPER LP RP SEMI stmt_lsit RCBR – כאשר השני מאפשר קריאת super אך רק כהצהרה ראשונה.

שלב ה-semantics: אין כיוון שטופל ב-parser.

שלב ה-LIR:

בנוסף למה שעשינו ב-א':

שמים את ה-DV של המחלקה ה-super ברגיסטר, מחשבים את ה-offset של ה-constructor ואז מבצעים VirtualCall על ה-constructor של מחלקת ה-super.

שלב ה-Assembly: אין תוספות.

ג.

שלב ה-parser: להוסיף בגזירת קריאה למתודה גזירה נוספת של חוק מהצורה:
… SUPER DOT ID:method_name LP (המשך כמו למתודה רגילה).

שלב ה-semantics:

- בודקים שלמחלקה יש מחלקת אב.
- בודקים שלאבא יש מתודה בשם הנכון
- ה-type הוא ה-type של מתודת האב.
- בדיקת types: בודקים התאמת ארגומנטים כמו ב-call רגיל.

שלב תרגום ה-LIR: כמו בסעיף קודם לוקחים את ה-DV של מחלקת האב move DV_A R1, מחפשים ב-ClassLayout של האב (מוצאים אותו לפי שם המחלקה) ואז מוסיפים קוד כמו קריאה וירטואלית (יוצרים ASTNode חדש מסוג קריאה וירטואלית עם הנתונים המתאימים ומבצעים עליו visit עם ה-visitor שאחראי על יצירת הקוד – קיצור דרך).
שלב התרגום ל-Assembly: אין שינויים, הכל נגזר מהשינויים ב-LIR.

שאלה 4 (15 נק') : הקצאת אוגרים

כזכור, אלגוריתם Sethi-Ullman לעצי ביטויים מתבצע בשני שלבים – שלב ה-(Bottom-Up(Labeling ושלב ה- Top Down, כאשר בשלב ה –Top Down מבקרים בתתי עצים כבדים יותר תחילה.

א. (10 נק') אנו רוצים לשנות את אלגוריתם Sethi-Ullman כך שבשלב ה Top Down נעדיף לבקר בתתי עצים קלים יותר תחילה. מה יהיה מספר הרגיסטרים המוקצים ב-best case של האלגוריתם לאחר השינוי. **רמז:** היעזר ביחס בין מספר האוגרים לגובה עץ הביטוי.

היזכר באלגוריתם הקצאת האוגרים הגלובאלי ע"י צביעת גרף ההפרעות.

ב. (5 נק') נתונה מכונה עם אוגרים במהירויות גישה שונות - לשם פשטות הנח קבוצת אוגרים "רגילים" וקבוצת אוגרים "מהירים". כיצד תשנה את אלגוריתם ההקצאה הגלובלית כך שהקוד הנוצר יהיה יעיל ככל האפשר. אילו שלבים באלגוריתם יושפעו.

פתרון:
א. למעשה השינוי במימוש לא משנה את הביצועים ב-best case. במקרה שהצמתים מסודרים בשרשרת אחד אחרי השני נחוץ רק רגיסטר 1.

ב. נחזיק 2 גרפים: גרף אוגרים מהירים וגרף אוגרים איטיים.

האלגוריתם יעבוד כרגיל, עם גרף האוגרים המהירים בלבד. כאשר מגיע למצב actual spill, ינסה להצמיד את המשתנה לגרף האוגרים האיטיים (תחת ההגבלות) – אם הצליח – יופי, אם נכשל, נבצע spill אמיתי לזיכרון.

בהצלחה!
רן ואוהד

תאריך: 21.02.2007
מועד: א'
מרצה: פרופ' מולי שגיב
מתרגל: רומן מנביץ'
חומר: פתוח
משך: 3.5 שעות

מבחן בקומפילציה

שאלה 1 (25 נקודות) : סיווג מאורעות

סווגי את המאורעות הבאים לפי זמן ריצה, זמן קומפילציה, או זמן בניית הקומפיילר. פרטי **מתי בדיוק מתרחש כל מאורע ואילו נתונים נדרשים** אליו על-ידי הקומפיילר או כותבת הקומפיילר, כלומר באילו מבני נתונים ואלגוריתמים משתמשים לצורך המאורע. במידה ויש מספר תשובות נכונות, הסבירי את כולן:

א. (5 נק') נמצא כי אובייקט o אינו נגיש, ולכן ניתן להשתמש בזיכרון שהוקצה לו להקצאה אחרת.
ב. (5 נק') נמצא כי בביטוי x.f=f, ה-f מימין להשמה הינו משתנה מקומי וה-f משמאל להשמה הינו שדה של המחלקה A.
ג. (5 נק') נמצא כי גודל טבלת הפונקציות הוירטואליות (dispatch table) של אובייקטים מהמחלקה A הוא 16 בתים וגודל טבלת הפונקציות הוירטואליות של המחלקה B שיורשת מ-A הוא 20 בתים.
ד. (5 נק') התגלה כי הפרמטר הפורמאלי הראשון בעל טיפוס שונה מהפרמטר האקטואלי הראשון של פונקציה המוגדרת כ-extern (פונקציות המוכרזות כ-extern אינן מוגדרות ביחידת הקומפילציה הנוכחית).
ה. (5 נק') מתבצעת שמירה של ערך האוגר eax ברשומת ההפעלה.

פתרון:
א. זמן ריצה: ה-GC עובד בזמן ריצה ומוצא שאינו נגיש ומפנה את הזיכרון.
ב. זמן קומפילציה: בשלב ה-semantic checks לאחר שה-symbol tables בנויים, ה-visitor של בדיקות ההגדרות והטיפוסים בודק ב-assignment את מיקום הביטוי הימני והביטוי השמאלי (ואז בודק type checks).
ג. זמן קומפילציה: בשלב התרגום ל-LIR כאשר בונים את ה-ClassLayout נבנה ה-DV של כל class ואז ניתן לדעת את גודלו, ובשלב בניית קוד ה-Assembly בהקצאת מקום ל-DV של כל מחלקה מושכים את גודלו.
ד. משהו שקשור ללינקר.
ה. זמן ריצה: בזמן ריצה מתבצעת פעולת ההשמה עצמה ל-AR.

שאלה 2 (20 נקודות) : ניתוח תחבירי

נתון הדקדוק הבא להשוואת מסלולי זיכרון בתכנית ע"י פקודת assert:

$S \rightarrow$ assert C
$S \rightarrow$ assert lp C rp
$C \rightarrow P$ eq P
$P \rightarrow$ id | id dot P

ה-tokens המופיעים בדקדוק מייצגים את המחרוזות המתאימות: dot='.' eq:'==' rp:')' lp:'('.

דוגמאות לקלטים חוקיים הם הקלט "assert(x.a==y.b.c)" והקלט "assert x.n.n==y".
תזכורת: בשיטת LL(k) אלגוריתם הניתוח משתמש בטבלה הממפה כל k סימנים מהקלט (סדרה של טרמינלים ומשתני דקדוק) לכלל הגזירה שבו יש להשתמש כדי להחליף את משתנה הגזירה השמאלי ביותר.

א. (3 נק') מהו ערך של k הדרוש על-מנת לבצע ניתוח תחבירי לדקדוק הנתון בשיטת LL(k)? (כמה tokens קדימה צריך להסתכל כדי להחליט באיזה כלל גזירה לבחור?) הסבירי.
ב. (3 נק') האם ניתן להפחית את ערך k ע"י שינוי הדקדוק? אם כן, הראי כיצד. אם לא, הסבירי למה לא ניתן לעשות זאת.
ג. (4 נק') הראי שתת הדקדוק הניתן על-ידי כללי הגזירה של P (שני הכללים האחרונים) אינו שייך ל-LR(0).
ד. (10 נק') הראי כיצד ניתן לשכתב אותו כך שישתייך ל-LR(0). **רמז:** השתמשי בכלל החדש $P' \rightarrow id$.
בני את דיאגרמת המצבים (האוטומט) של הדקדוק המשוכתב ואת טבלת הניתוח, והראי כיצד מתקבל הביטוי "x.n.n".

פתרון:
ג. הדקדוק החדש:

$S \rightarrow P\$$
$P \rightarrow P\ dot\ P' \mid P''$
$P' \rightarrow id$

שאלה 3 (35 נקודות) : הרחבה לשפה ורשומות הפעלה

בשאלה זו הינך מתבקשת להסביר את השינויים הדרושים בקומפיילר, כדי להוסיף לשפת IC פקודה המאפשרת הקצאת מערכים בתוך רשומת ההפעלה הנוכחית, בדומה לפונקצית alloca שבשפת C. התחביר של ביטוי הקצאת מערך נתון ע"י כלל הגזירה הבא:

STACK_ARRAY_ALLOC → alloca TYPE '[' EXPR ']'

כאשר TYPE הינו סוג האלמנטים המאוחסנים במערך ו-EXPR הינו ביטוי המתאר את מספר האלמנטים שבמערך (התחביר זהה להקצאת מערך ע"י new, מלבד השימוש במילה alloca במקום המילה new).

בשונה ממערכים המוקצים בזיכרון הערמה (heap) ומנוהלים ע"י מנגנון ה-garbage collection, הזיכרון המוקצה למערכים בתוך רשומת ההפעלה משתחרר כאשר הפונקציה שבה המערך הוקצה חוזרת מהקריאה. לכן, נוספת דרישת הנכונות הבאה על השימוש בביטויי alloca – דרישה שאינה ניתנת לבדיקה בזמן קומפילציה, במקרה הכללי.

דרישה: פניה למערך שהוקצה ע"י alloca מותר אך ורק כל עוד הקריאה לפונקציה שבה הוא הוקצה לא חזרה.

לדוגמא, הביטוי alloca int[size] בתכנית הרשומה למטה הינו ביטוי הקצאה חוקי. כאשר הוא מתבצע, רשומת ההפעלה של הפונקציה bar גדלה ב-16 בתים. כאשר הקריאה לפונקציה bar חוזרת, רשומת ההפעלה של bar יוצאת ממחסנית הקריאות והזיכרון המוקצה למערך arr נאסף חזרה, כלומר נחשב כאינו מוקצה לשימוש התכנית ופנייה לאבריו נחשבת לא-חוקית.

```
class A {
  static void main(string[] args) {
    A.bar(3);
  }
  static void bar(int size) {
    int[] arr = alloca int[size];
    A.fill(arr, 7);
  }
  static void fill(int[] nums, int val) {
    int i = 0;
    while (i < nums.length) {
      nums[i] = val;
      i = i + 1;
    }
  }
}
```

עני על הסעיפים הבאים ושימי לב:

- הקצאת המערך אינה נעשית ע"י פונקצית ספרייה אלא ע"י המימוש המוצע על-ידך.
- בדיקת דרישת הנכונות מהווה בעיה בלתי-כריעה, ולכן אינה ניתנת לבדיקה בזמן קומפילציה. אינך נדרשת להתייחס לכך בפתרון סעיפים א', ב', ו-ג'.

א. (5 נק') הסבירי למה צריך את דרישת הנכונות ותני דוגמא לתכנית שבה הדרישה אינה מתקיימת ויוצרת בעיה.

ב. (10 נק') הסבירי את השינויים הנדרשים בשלבים השונים של הקומפיילר תוך-כדי מתן דגש לפרטים הבאים:

- השינויים הדרושים בניהול רשומות ההפעלה,
- מימוש ביטויי alloca,
- תמיכה באופרטור length,
- ביצוע פקודות פנייה לאברי מערך המוקצה ע"י alloca,
- בדיקות המתבצעות בזמן קומפילציה ובדיקות המתבצעות בזמן ריצה.

ב. (10 נק') הראי את קוד האסמבלי שנוצר ע"י הקומפיילר לפונקציה bar. נא הוסיפי הערות לצד הקוד.
לשימושך, הדף האחרון בבחינה מכיל סיכום הוראות אסמבלי x86 ודוגמאות.

ג. (10 נק') הראי את מצב הזיכרון הקיים בתכנית הדוגמא בנקודות הבאות בתכנית:

1. לפני ביצוע הפקודה הראשונה בפונקציה bar,
2. לאחר ביצוע הפקודה,
3. בנקודת החזרה של הפונקציה fill,
4. בנקודת החזרה של הפונקציה bar.

הניחי שבתחילת ביצוע הפונקציה foo, כתובת מחסנית הקריאות הינה 1000 ושמבנה רשומות ההפעלה הוא כמו שנלמד בשיעור ובתרגול, כלומר המחסנית מתחילה בכתובות גבוהות ומתקדמת לכתובות הנמוכות בכל קריאה. כמו-כן, הניחי שערכי האוגרים לא נשמרים ברשומת ההפעלה.

התרשים הבא מראה את מצב הזיכרון לאחר העברת הפרמטר האקטואלי בקריאה A.bar(3), לפני ביצוע הקריאה.

	address	value
	1000	args
	996	main return address
FP →	992	previous fp
SP →	988	3 (argument to bar)
	…	…

שאלת בונוס: (5 נק')

הציעי דרך לבדוק את דרישת הנכונות בדרך כלשהי. לדוגמא, בזמן קומפילציה באופן קונסרבטיבי (כלומר בדיקה ש**תמיד** מתריעה על תכניות אשר מפרות את דרישת הנכונות ולפעמים גם מתריעה על תכניות חוקיות, אך מעט ככל האפשר),
או בעזרת בדיקות בזמן ריצה, או שילוב כלשהו.

פתרון:

א. הדרישה נצרכת אחרת יכול להיווצר מצב בו array מוגדר מחוץ ל-bar, בתוך ה-bar מוקצה מקום על ה-stack, אך כאשר יוצאים מ-bar תוכן המערך מכיל זבל והמערך עדיין נגיש. כעת אם תהיה מתודה חדשה foo כלשהי שתנסה לגשת לאיבר במערך, היא למעשה תנסה לגשת למקום ב-stack שהוא חלק מה-AR של foo עצמה, וזה מסוכן.

ב. שינויים הנדרשים בשלבי הקומפיילר:

בנוסף להוספת TOKEN ל-ALLOCA והוספת הכלל לעיל ל-parser, נוסיף ASTNode ל-AllocaCall וטיפול בו ב-type table ובדיקות סמנטיות:

- לבדוק שה-type של ה-size הוא int ואינו שלילי
- בדיקת type דומה לבדיקת type ב-NewArray.

בשלב ה-LIR: מוסיפים מתודת allocaArray שמקבלת size כמספר האלמנטים במערך.

בשלב ה-code generation:

- הקריאה למתודה מורידה את ה-sp ל(מספר האיברים במערך + 1)*4
- ה-ret יהיה sp+4, ולמקום sp יוכנס גודל המערך (מספר האיברים).

בחזרה מפונקציית ה-allocaArray ה-ret, שהוא מצביע לתחילת המערך, יוכנס לתוך ה-temporary שמחזיק את המערך.
גישה למערך והוצאת length יהיו בדיוק כמו קודם: ל-length ניגש לכתובת arr-4 ולאיבר i ניגש לכתובת arr+4i.

בדיקות זמן ריצה:

- Null pointer reference
- Array bound

ב.

שאלה 4 (20 נק') : הקצאת אוגרים גלובאלית

תני דוגמא לגרף הפרעות (interference graph) עבורו conservative coalescing מוביל לפקודות move מיותרות, כלומר ניתן ליצר קוד עם שימוש באותו מספר של אוגרים (ללא כתיבות\קריאות נוספות), אשר מבטל יותר פקודות move.

הדרכה:

1. (10 נק') הראי את גרף ההפרעות (אין חובה לרשום את התכנית), והדגימי את פעולת האלגוריתם הצביעה שנלמד בכתה במלואו על הגרף למכונה בעלת שני אוגרים (אפשר עם יותר אוגרים אם לא מצליחים עם שניים).
2. (10 נק') תני דוגמא לצביעה חוקית של הגרף עבור מספר הרגיסטרים שבחרת בסעף 1, שבה יש פחות פעולות move מאשר מספר הפעולות המתקבל מהאלגוריתם בסעיף הקודם.

להזכירך, אלגוריתם ה-graph coloring, כפי שנלמד בכיתה בוחר את הרגיסטר לצורך spill ע"י יוריסטיקה המשתמשת ב-priority המוגדר כך:

priority = (uo + 10 * ui) / deg
uo = use + def outside of the loop
ui = use + def within the loop
deg = degree in the interference graph

פתרון:

נסתכל על גרף בו:

- r1,r2 מחוברים בקשת מלאה
- a מחובר בקשת מקווקוות ל-r1,r2,b
- b מחובר בקשת מקווקוות ל-r1,a ובקשת מלאה ל-r2

בגרף כזה, אם ב-coalescing הראשון יבחר a ויאוחד עם r1, אז ב-coalescing השני b יאוחד עם r1 גם כן. לעומת זאת, אם תחילה a יאוחד עם r2, תחסך פעולת move מיותרת לעומת המקרה הראשון. קוד לדוגמא עליו נבצע את האלגוריתם ונגיע לתשובה:

```
Mov 4 a /* a
Mov a r1/* r1
Print r1 /* r1
Mov 5 b /* b, r1
Mov b r2 /* r1,r2
Compare r1 r2 /* -
```

תאריך: 14.10.2007
מועד: ב'
מרצה: פרופ' מולי שגיב
מתרגל: רומן מנביץ'
חומר: פתוח
משך: 3.5 שעות

מבחן בקומפילציה

לכל שאלה, נא לקרוא את כל סעיפיה לפני ניסיון לפתור אותה.

שאלה 1 (25 נקודות) : סיווג מאורעות

סווגי את המאורעות הבאים לפי זמן ריצה, זמן קומפילציה, או זמן בניית הקומפיילר. פרטי **מתי בדיוק מתרחש כל מאורע ואילו נתונים נדרשים** אליו על-ידי הקומפיילר או כותבת הקומפיילר, כלומר באילו מבני נתונים ואלגוריתמים משתמשים לצורך המאורע. במידה ויש מספר תשובות נכונות, הסבירי את כולן:

א. (5 נק') בדקדוק G ישנה רקורסיה שמאלית ולכן כדי להתאים את הדקדוק לשיטת הניתוח התחבירי הוחלט לבצע left-factoring (באיזו שיטת ניתוח תחבירי מדובר?).
ב. (5 נק') ההשמה a[i]=b[i] אינה חוקית בהינתן ההכרזות הבאות:
int[] a;
boolean[] b;
ג. (5 נק') כאשר ההכרזה על המשתנה המקומי x זוהתה כבר היה סימן (symbol) מתאים ל-x בטבלה.
ד. (5 נק') הוחלט שהאוגר ebx (register) נשמר אך ורק לצורך אחסון ערכו של המשתנה הזמני R1 בפונקציה foo.
ה. (5 נק') טבלת הפונקציות של המחלקה B כוללת את הפונקציות הבאות: [B_rise, A_shine].

פתרון:
א. זמן בניית הקומפיילר: הנתונים הנדרשים הם הדקדוק. שיטת הניתוח התחבירי במקרה זה היא LL.
ב. זמן קומפילציה: בשלב ה-semantic type check, משתמשים ב-symbol tables כדי לבדוק את ה-type של כל אחד מהערכים משני צידי ההשמה.
ג. זמן קומפילציה: בשלב בניית ה-symbol table לפני שכל משתנה מוכנס לטבלה בסקופ שלו, נבדק שלא קיים כמוהו.
ד. זמן קומפילציה: בשלב ה-code generation (assembly), משתמשים בגרף ההפרעות.
ה. זמן קומפילציה: בשלב התרגום ל-LIR בשלב יצירת ה-ClassLayout.

שאלה 2 (20 נקודות) : ניתוח תחבירי

נתונות השפות הבאות מעל התווים a ו-b :
$L1 = \{a^n b \mid n = 1,2,\ldots\}$ -ו
$L2 = \{a^n b \mid n = 1,2,\ldots\} \cup \{a^n c \mid n = 1,2,\ldots\}$.

א. (6 נק') נסחי דקדוק חסר-הקשר לשפה L1 ששייך ל-LL(k) והראי את הגזירה של המחרוזת aab.
תזכורת: דקדוק LL(k) גוזר מחרוזת בצורה הבאה: רושמים את סימן הדקדוק הראשי ובאיטרציה מחליפים את סימן הדקדוק השמאלי ביותר בכלל דקדוק הנקבע ע"י k התווים הבאים בקלט (ע"י טבלא מתאימה), עד אשר מתקבל מחרוזת ללא סימני דקדוק (רק תווים).
ב. (7 נק') נסחי דקדוק לשפה L2 והראי שהוא ב-LR(0) ע"י בניית דיאגרמת מצבים וטבלת ניתוח.
ג. (7 נק') הפעילי את הנתח שבנית בסעיף הקודם על הקלט aac.

פתרון:
ב. להלן דקדוק ל-L2:

$S \rightarrow A\$$
$A \rightarrow aA \mid ab \mid ac$

שאלה 3 (30 נקודות) : הרחבות לשפת IC

בשאלה זו הינך מתבקשת להסביר אילו שינויים דרושים בקומפיילר לשפת IC על-מנת לתמוך בפרמטרים מסוג by reference, כפי שמוסבר ומודגם להלן ע"י תכנית הדוגמא הבאה.

```
class ByRefExample {
  static void main(string [] args) {
    int x = 5;
    increment(x);
    Library.println(x);
  }

  static void increment(int & y) {
    y = y + 1;
  }
}
```

פרמטר מוכרז כ-reference parameter ע"י הוספת הסימן `&` לאחר הכרזת הסוג שלו. למשל, בתכנית הדוגמא, הפרמטר `y` הינו reference parameter והארגומנט `x` מועבר by reference, כך שהפונקציה `increment` משנה למעשה את הערך של המשתנה המקומי `x` של הפונקציה `main`. התכנית מדפיסה למסך את המספר 6.

א. (7 נק') פרטי את השינויים הדרושים בשלבים הבאים כדי לתמוך בסוג הפרמטר החדש: ניתוח לקסיקאלי, ניתוח תחבירי ובניית AST, בניית טבלת הסוגים ובדיקות סמנטיות (כולל type-checking).
בשלב הבדיקות הסמנטיות, נא התייחסי בפרט לבדיקת חוקיות הארגומנטים המועברים לפרמטרים by reference.

ב. (2 נק') תני דוגמא לשורת קוד במקום הקריאה `increment(x)` שבתכנית הדוגמא, אשר מפרה את הבדיקות הסמנטיות החדשות להעברת ארגומנטים by reference, שפירטת בסעיף הקודם.

ג. (7 נק') פרטי את מימוש העברת הפרמטרים ברמת ה-LIR וקוד האסמבלי הנוצר.

ד. (7 נק') הראי את קוד האסמבלי שנוצר לתכנית הדוגמא ברמת פירוט מספקת רק כדי להראות את אופן העברת `x` לפונקציה `increment` ואת ביצוע הפקודה `y = y + 1`. נא הוסיפי הערות לצד הקוד.
(ניתן להתעלם מהשורה `Library.println(x)`.)
לשימושך, הדף האחרון בבחינה מכיל סיכום הוראות אסמבלי x86 ודוגמאות שימוש.

ה. (7 נק') השתמשי בתכנית שבנית בסעיף הקודם כדי לחשב את מצב המחסנית לאחר ביצוע הפקודה `y = y + 1`. הראי את מצב המחסנית וצייני את משמעות הנתונים במחסנית.

פתרון:

א. השינויים הנדרשים:
ניתוח לקסיקלי: הוספת TOKEN מסוג AMPERSAND
ניתוח תחבירי: הוספת AMPERSAND אופציונאלי לפני כל FORMAL בכללי הגזירה, וב-AST נוסיף ל-Formal משתנה isReference.
בניית ה-type table: תלוי איך מסתכלים על override: אם השוני של החתימה במתודה הוא שקובע את הטיפוס שלה, נקבע טיפוס חדש מסוג מצביע ל(טיפוס כלשהו). אם השוני יוגדר לפי הקריאה, כיוון שהקריאה היא זהה עם מצביע או בלי מצביע (בניגוד ל-C למשל), לא נעשה type חדש ל-reference ונתייחס לשתי המתודות כבעלות אותו type. נבחר למימוש זה באופציה השניה.
בדיקות סמנטיות: צריך לבדוק שהפרמטר שנשלח הוא variable location או array location.

ב. דוגמא לקריאה לא חוקית: increment(3)

ג. מימוש ב-LIR:
- מוסיפים בכל קריאה למתודה: אם פרמטר נשלח by reference הקריאה תהיה מהצורה foo(R1=&x) – כלומר הוספת הסימן &. מאוחר יותר ב-code generation זה יסמן העברה של מצביע ולא של ערך.
- בתחילת מתודה שמים במשתנה מקומי את הערך היושב במצביע שהתקבל, ומשתמשים בערך זה כערך רגיל.
- בסוף המתודה, שמים את הערך הסופי למשתנה המקומי לעיל לתוך התוכן של המצביע שקיבלנו כקלט. כך ישתנה המשתנה המקורי ממש.

מימוש ב-assembly:
- בכל קריאה שיש לרפרנס נדחוף למחסנית במקום למשל ebx% נשלח את (%ebx)
- ...

ד. ...

שאלה 4 (25 נק') : הקצאת אוגרים גלובאלית

להלן תוכנית בשפת ביניים ונתוני החיות של המשתנים המופיעים בה (תוצאת ההפעלה של אלגוריתם ה-liveness analysis שנלמד בכיתה). שימי-לב שנתוני החיות מתייחסים למצב אחרי ביצוע הפקודה המתאימה.

```
enter:
        x := 8;  /* x, r1, r2 */
        y := r1; /* x, y, r1 */
        z := r2; /* x, y, z */
loop:
        x := x - 1; /* x, y, z */
        y := y + z; /* x, y */
        z := 2;     /* x, y */
        if x > 0 goto loop;
        r1 := y;    /* r1 */
        return;     /* r1 */
```

פתרון:
נשים לב כי אין potential spill ולכן אין צורך להשתמש בחישוב Priority כלל.
כתיבת הגרף פשוטה. הצמתים יוכנסו ל-stack באופן הבא:
z,y,x כאשר x בראש המחסנית.
ב-pop כל אחד מהמשתנים לעיל יוכנסו באופן יחיד לתוך כל אחד מהרגיסטרים (שאלה מאוד קלה).

א. (10 נק') בני את גרף ההפרעות (ה-interference graph) המתאים לנתוני החיות של המשתנים ולתכנית.

ב. (15 נק') הפעילי את אלגוריתם ה-graph coloring על גרף ההפרעות מהסעיף הקודם והניחי שבמכונה יש בדיוק שלושה רגיסטרים r1, r2 ו-r3, כאשר r1 ו-r2 המופיעים בקוד מתאימים לרגיסטרים בעלי אותו שם.
כמו-כן השתמשי בקריטריון של Briggs לביצוע איחוד צמתים (coalescing): שני צמתים a ו-b ניתנים לאיחוד אם לצומת המאוחד יהיו פחות מ-k שכנים בעלי דרגה גדולה או שווה ל-k (בשאלה הנוכחית k=3).

להזכירך, אלגוריתם ה-graph coloring, כפי שנלמד בכיתה בוחר את הרגיסטר לצורך spill ע"י יוריסטיקה המשתמשת ב-priority המוגדר כך:

priority = (uo + 10 * ui) / deg
uo = use + def outside of the loop
ui = use + def within the loop
deg = degree in the interference graph