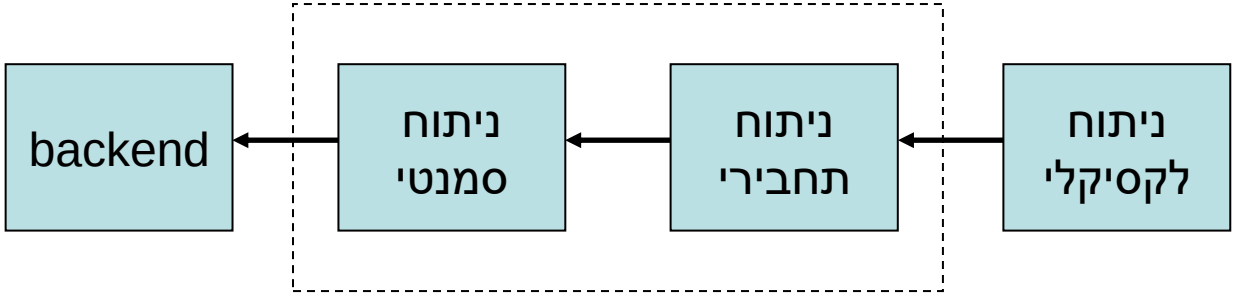
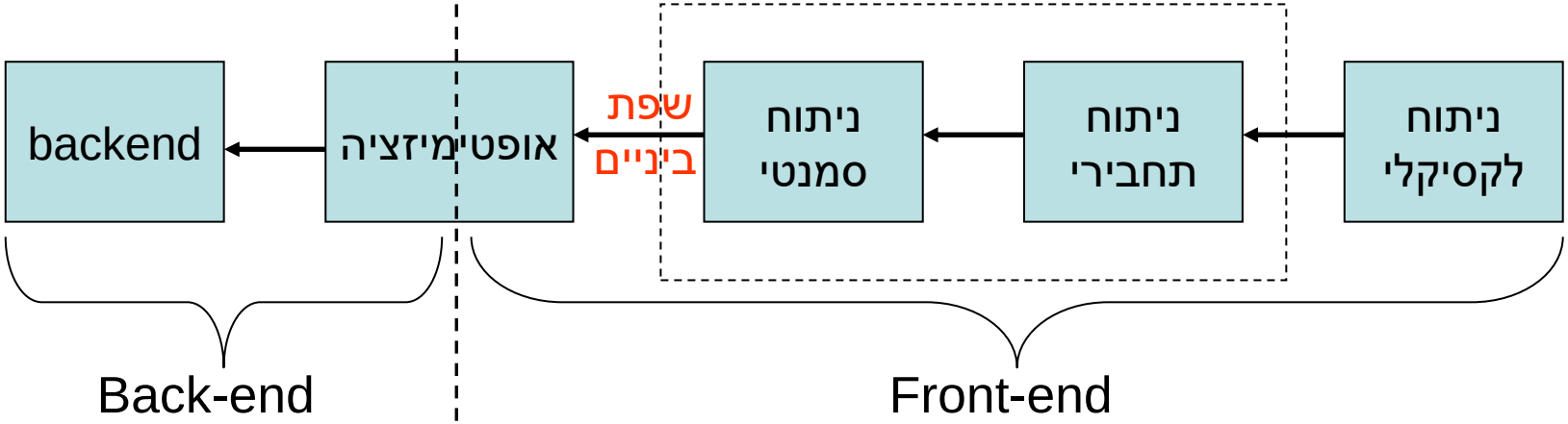
תרגום לקוד ביניים

מבנה סכמתי של קומפיילר

- עד כה ראינו:



- בפועל:



שפת ביניים

- מדוע לא לתרגם ישר לשפת היעד?
  - פיתוח מהיר יותר של קומפיילר למערכת חדשה
    - נדרש לכתוב מחדש רק את ה Back-end.
  - פיתוח אופטימיזציות כלליות.

- שפת ביניים איתה נעבוד בקורס: שפת הרביעיות
  - דומה מאד לשפת אסמבלר.
  - סדרה של פקודות מהצורה x := y OP z

שפת ביניים (המשך)

- נעבוד עם 4 סוגי פקודות בלבד:

1. פעולה אריתמטית: $t_1 := t_2 + t_3$
2. קפיצה לא-מותנית: goto label
3. קפיצה מותנית: if t_1 relop t_2 goto label
4. תווית: label:

- קיימות פקודות רבות נוספות....

דוגמה

$$a = b + c + d$$

$$\Downarrow$$

$$t_1 := b + c$$

$$t_2 := t_1 + d$$

$$a := t_2$$

- מדוע צריך את t_2? לא צריך (אופטימיזציה)

טעויות נפוצות

- דברים שאינם קוד ביניים:
- if … else …
  - בשפת ביניים אין else.
- if (x > 1 && y > 1) goto …
  - התנאי בקפיצה אינו מורכב.
    - נראה ייצוג חלופי בשפת ביניים לשני המקרים הללו.
- x = 1000

  goto x
  - ניתן לקפוץ רק לתוויות קבועות.

פונקציות לפריסת קוד

- נצטרך את התכונות הסמנטיות הבאות:
  - code: מחזיקה את הקוד הנוצר.
  - var: עבור ביטויים אריתמטיים זהו המשתנה הזמני שהוקצה לביטוי (למשל: כתובת יחסית במחסנית...)

פונקציות לפריסת קוד

- פונקציות עזר:
  - freshVar(): יוצרת משתנה זמני חדש (בעל מזהה ייחודי), כולל הכנסתו לsymbol table.
  - freshLabel(): מחזירה תווית חדשה.
  - gen(string) : מקבלת מחרוזת שמייצגת פקודות ומייצרת את הפקודות בשפת הביניים.

שיטות לייצור קוד

נציג 2 שיטות לייצור קוד:

<u>שיטה ראשונה:</u> הקוד נאגר בתכונה סמנטית בשם code:

- לכל משתנה בדקדוק יש תכונה סמנטית בשם code, אשר אוגרת את הקוד שנוצר מהמשתנים שכבר נגזרו.
- בסוף הניתוח התכונה code של המשתנה התחילי מכילה את כל הקוד של התוכנית.

שיטה ראשונה ליצירת קוד - דוגמה

- דוגמה: תרגום ביטויים אריתמטיים
  - נגדיר למשתנה E את התכונות code ו-var

(var יכיל את המשתנה הזמני אליו נשמר ערך הביטוי).

$E \rightarrow E_1 + E_2$ {

E.var = freshVar();

E.code = E_1.code || E_2.code

|| gen (E.var ‘:=‘ E_1.var ‘+’ E_2.var); }

יצירת מקום לתוצאה

יצירת הקוד:

$t_1 \leftarrow E_1$

$t_2 \leftarrow E_2$

$t_3 := t_1 + t_2$

שיטה שנייה ליצירת קוד

נחזיק buffer גלובלי שיכיל את כל הקוד שנוצר עד כה.

- קוד חדש יודפס ישירות לתוך הbuffer בעת ביצוע reduce לכלל.
- בעת סיום הגזירה, הbuffer יכיל את כל הקוד שנוצר.

שיטה שנייה ליצירת קוד - דוגמה

- דוגמה: תרגום ביטויים אריתמטיים
  - ל-E נגדיר תכונה var (אין צורך בתכונת code).
  - הפונקציה emit מדפיסה פקודה ל-buffer.

$E \rightarrow E_1 + E_2$ {

E.var = freshVar();

emit(E.var || ":=" || E_1.var || "+" || E_2.var); }

- היכן הקוד של E_1 ו-E_2?

השוואה בין שתי השיטות

- בשיטה 1 ניתן לקבוע את הסדר של קטעי הקוד ובשיטה 2 – לא ניתן
  - למשל: E.code = E_2.code || E_1.code || E.var || “:=“ || E1.var || “+” || E2.var;
  - במקום: E.code = E_1.code || E_2.code || E.var || “:=“ || E1.var || “+” || E2.var;
- שיטה 2 פשוטה ומהירה יותר
  - אין שרשורי קוד.

תרגום פשוט - הגדרה

- לכל קטע קוד (המתאים למשתנה גזירה מסויים) נקודת כניסה אחת (בראשיתו) ונקודת יציאה בסופו.
- התוצאה:
  - בתום ביצוע קטע קוד, מבוצע קטע הקוד המשורשר אחריו.
- עבור קוד ביניים המקיים תכונה זו:
  - ניתן לכתוב תרגום מונחה תחביר בו **אין צורך להעביר מידע על תוויות.**
  - עובדה זו מפשטת את התרגום.

הקושי

- כאשר מתרגמים מבנה בקרה, יעדי הקפיצות לא תמיד ידועים.
- לדוגמה: if B then S_1 else S_2
- לכאורה, יש יותר מנקודת יציאה אחת:
  - יש לבצע קטע קוד אחד במקרה והתנאי הבוליאני מתקיים,
  - או קטע קוד אחר במקרה והתנאי אינו מתקיים.
- איך המנתח יידע את מי משני קטעי הקוד עליו למקם מיד אחרי קטע הקוד לבדיקת התנאי B?

פתרון לבעיית if B then S_1 else S_2

```
if some_expression then
    some_code_1
else
    some_code_2
```

קטע הקוד השקול בשפת ביניים, לאחר תרגום פשוט:

```
code that evaluates some_expression into t13
    if  t13==0 goto L54
code for some_code_1
    goto L55
L54:
code for some_code_2
L55:
```

- קטע הקוד של בדיקת התנאי B יופיע בהתחלה ויחשב את התנאי לתוך משתנה זמני (לקטע קוד זה יציאה אחת בסופו).
- קטע הקוד שאחריו ימשיך לביצוע של S1 או S2 על פי ערכו של המשתנה הזמני.

שלב 1: הקוד לשיערוך הביטוי הבוליאני

```
E  ->  E1 '>' E2
{   tmplabel = freshLabel();
    E.var = freshVar();
    E.code = E1.code || E2.code
         || gen(E.var '= 0')
         || gen('if' E1.var '<=' E2.var 'goto' tmplabel)
         || gen(E.var '= 1')
         || gen(tmplabel ':'); }
```

שלב 2: תרגום פשוט עבור פקודת תנאי

```
S -> if E then S1 else S2
{   falseLabel = freshLabel(),
    endLabel = freshLabel();
    S.code = E.code
                || gen ('if' E.var '== 0' 'goto' falseLabel)
                || S1.code
                || gen ('goto' endLabel)
                || gen (falseLabel ':') || S2.code
                || gen (endLabel ':'); }
```

תרגום פשוט - חסרונות

- לא תמיד אפשר (או נוח) לדאוג ליציאה אחת מכל קטע קוד
  - Break (בלולאה).
  - Switch.
- פתרון: תכונות נוצרות שמייצגות תוויות.

דוגמה – תרגום משפט if מורכב

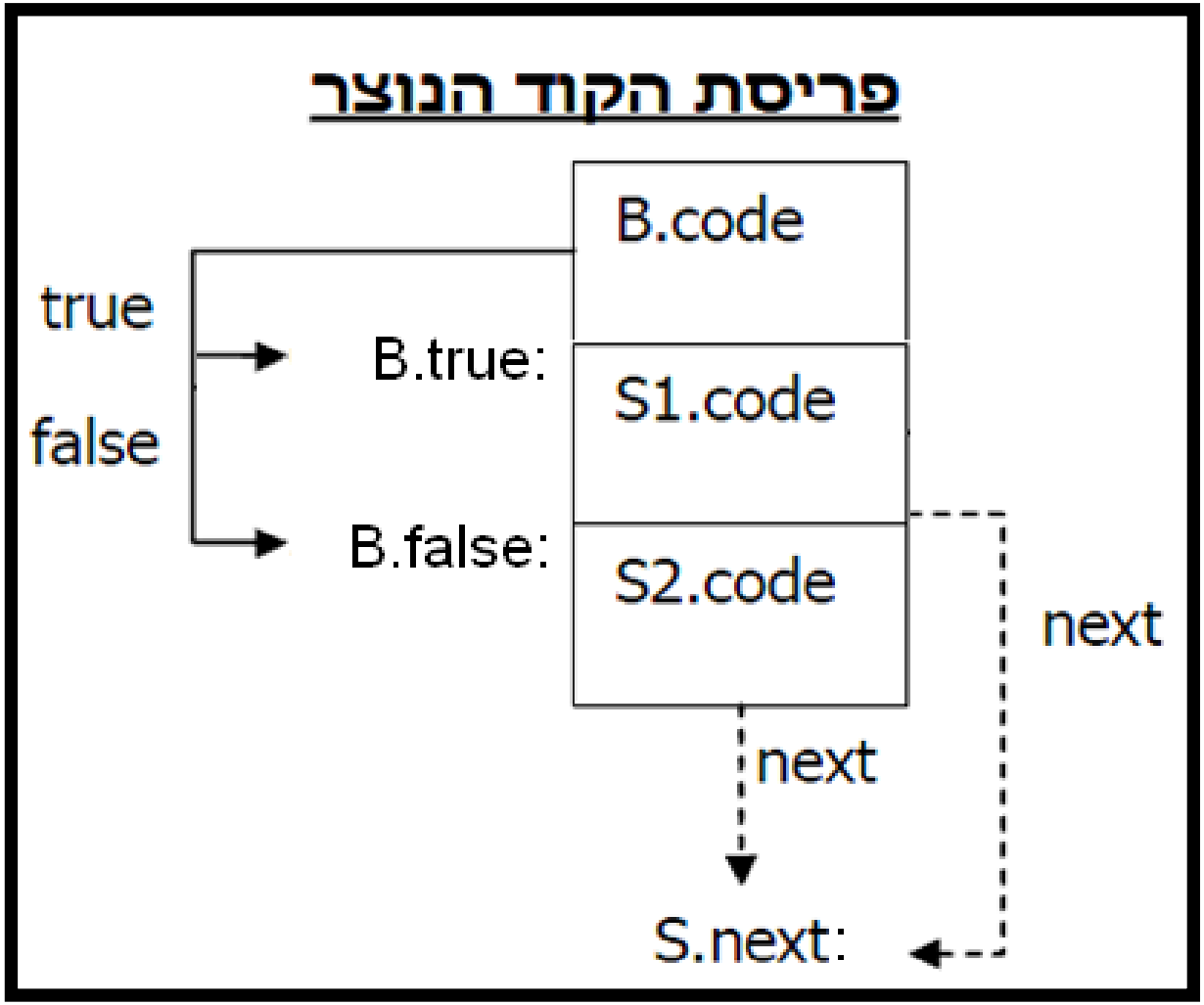
$$S \rightarrow \underline{\text{if}}\ B\ \underline{\text{then}}\ S_1\ \underline{\text{else}}\ S_2$$
$$B \rightarrow B_1\ \underline{\text{or}}\ B_2$$
$$B \rightarrow E_1\ \underline{\text{'>'}}\ E_2$$

התכונות (הנוצרות):

- B.false : התווית אליה תעבור התוכנית במקרה שערך הביטוי הוא false.
- B.true: התווית אליה תעבור התוכנית אם ערך הביטוי הוא true.
- S.next: התווית אליה נעבור בסיום הפקודה S.

דוגמה – תרגום משפט if מורכב

$S \rightarrow$ <u>if</u> B <u>then</u> S_1 <u>else</u> S_2
$B \rightarrow B_1$ <u>or</u> B_2
$B \rightarrow E_1$ <u>'>'</u> E_2



דוגמה – תרגום משפט if מורכב

<u>סכימת התרגום:</u>

S → <u>if</u> B <u>then</u> S_1 <u>else</u> S_2

{ S.next = freshLabel();

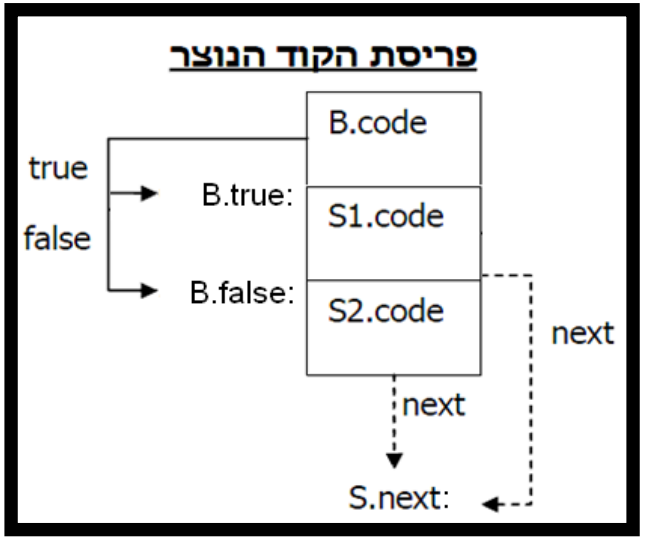
S.code = B.code

|| gen (**B.true** ':') || S1.code

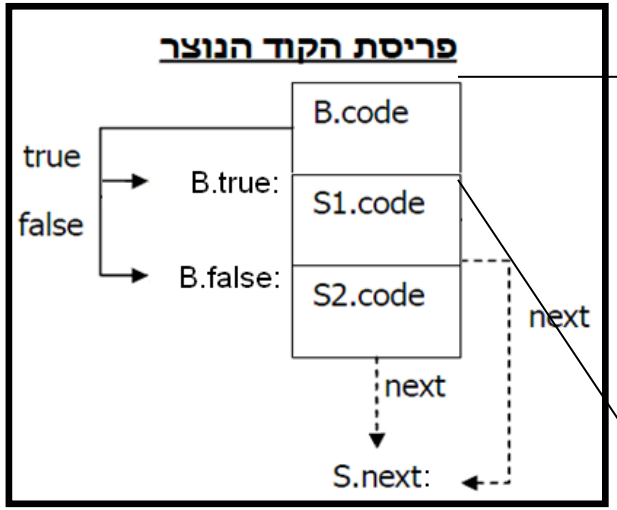
|| gen ('goto' **S.next**)

|| gen (**B.false** ':') || S2.code

|| gen (**S.next** ':'); }



המשך הדוגמה – הקוד עבור הביטוי הבוליאני



$B \rightarrow E_1$ '>' E_2

E_1.code

E_2.code

if (E_1.var > E_2.var) goto B.true
goto B.false

תזכורת: זהו המשתנה הזמני שהוקצה לביטוי (למשל מיקום במחסנית)

B → E1 '>' E2

{ B.true = freshLabel();

B.false = freshLabel();

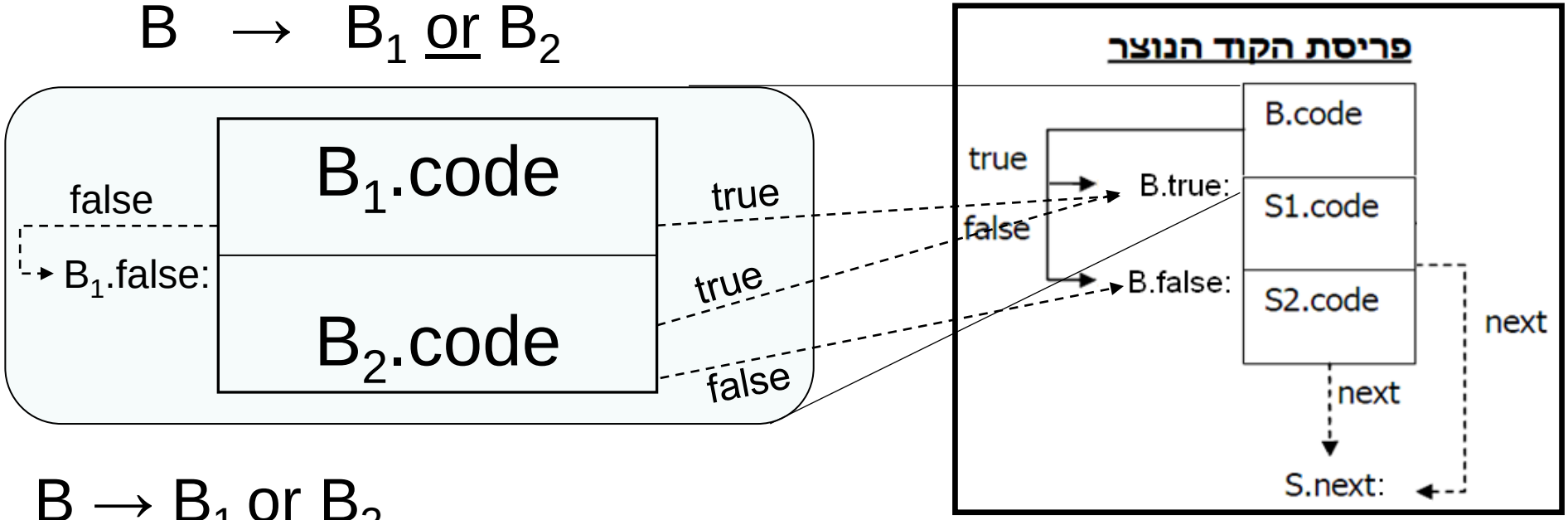
B.code = E1.code || E2.code

|| gen ('if' E1.var '>' E2.var 'goto' B.true)

|| gen ('goto' B.false) ; }

המשך הדוגמה – הקוד עבור הביטוי הבוליאני

$B \rightarrow B_1 \ \underline{or} \ B_2$



$B \rightarrow B_1 \ \underline{or} \ B_2$

{ B.true = B_2.true;

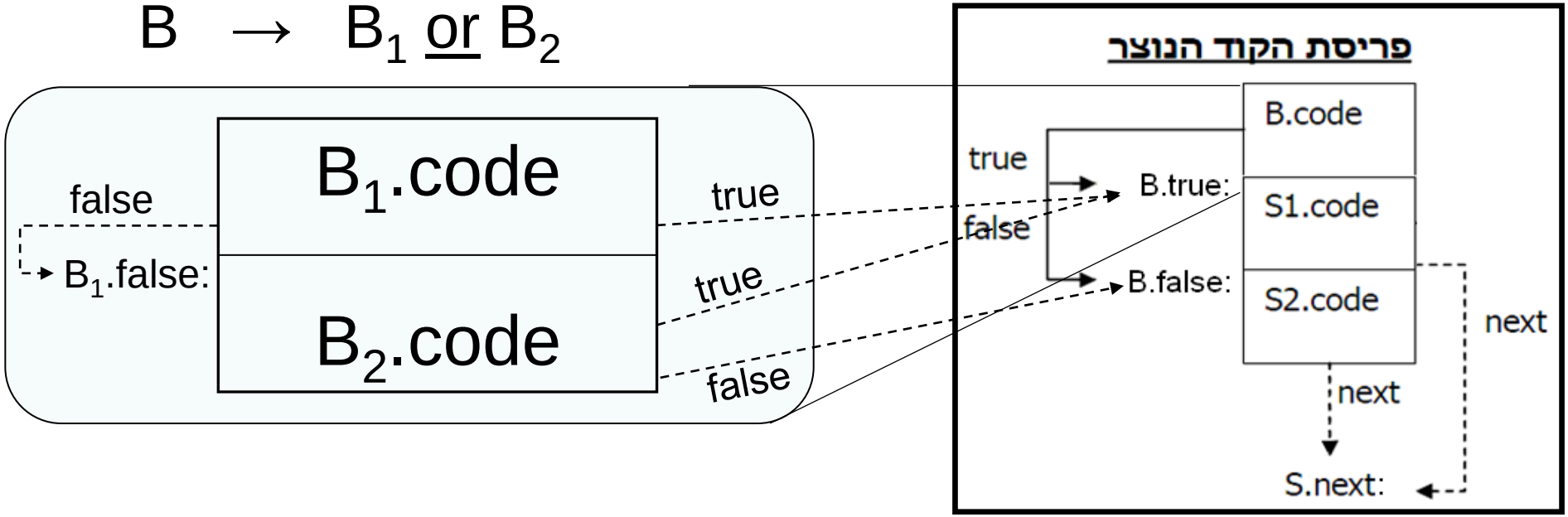
B.false = B_2.false;

B.code = B_1.code || gen (**B_1.false** ':') || B_2.code

|| gen(**B_1.true** ':') || gen('goto ' **B.true** ':'); }

המשך הדוגמה – הקוד עבור הביטוי הבוליאני

$B \rightarrow B_1 \ \underline{or} \ B_2$



הערות:

- הבדיקות מבוצעת בשיטת Short circuit lazy evaluation :
  - ברגע שהתשובה ידועה מפסיקים לחשב.
  - המשמעות היא כמו של האופרטור || ולא כמו האופרטור |

שרשור קפיצות

$B \rightarrow B_1 \ \underline{\text{or}} \ B_2$

הקוד שיווצר:

B_1.code

B_1.false:

B_2.code

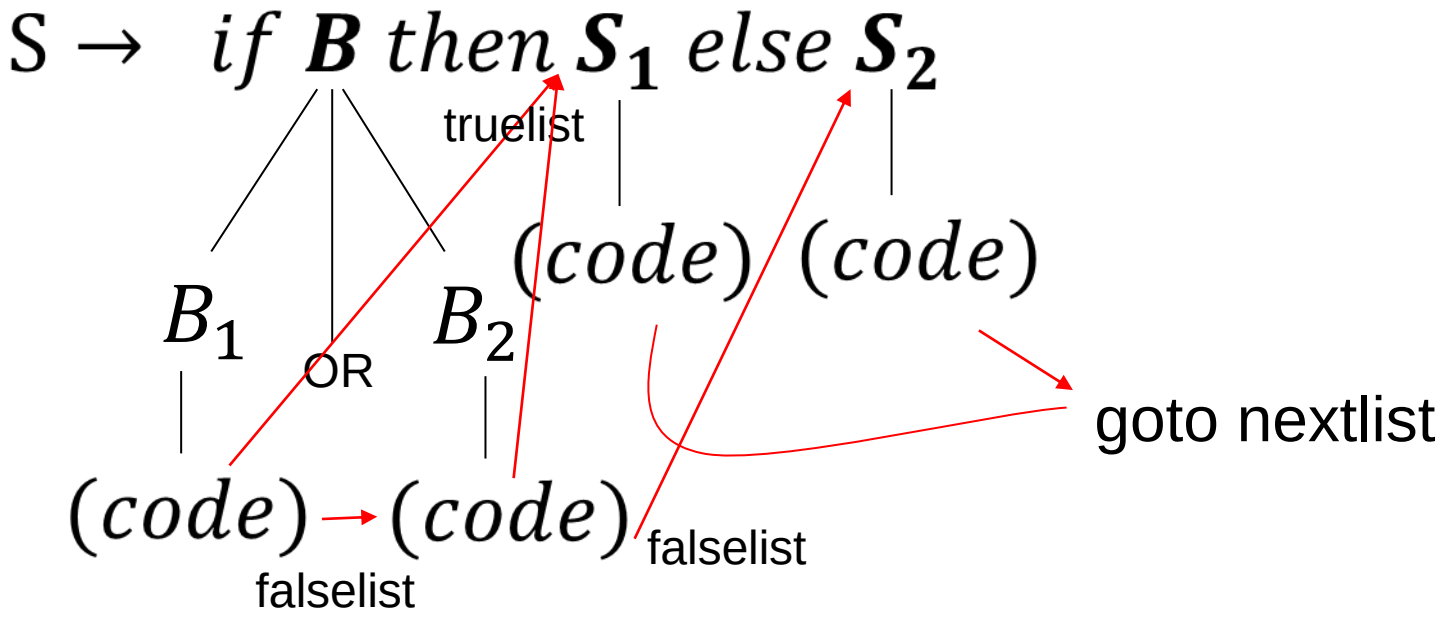
B_1.true:

goto **B.true**:

קפיצה מיותרת
אופטימזציה

עץ הגזירה

- יש צורך לחווט את היציאות מכל קטע קוד למקום הנכון.



תזכורת- גישה למחסנית

- בגרסאות חדשות של בייזון ניתן לגשת למחסנית מאמצע כלל:

```
A : B {printf("%d",$1);} C {printf("%d",$3);};
```

- איך כזה דבר מתורגם?

```
A : B M C {printf("%d",$3);};
M : { YYSTYPE b = stack.peek(0); //top
      printf("%d",b); };
```

- שימו לב! $$ באמצע כלל הוא $$ של המרקר!

פתרון ללא else

$S \rightarrow if\ \boldsymbol{B}\ \boldsymbol{M}\ then\ \boldsymbol{S_1}$

- נרצה **מרקר M** שמטרתו "יצירת לייבל" במיקום מסוים.

M → ε { emit(B.true || ‘:’); }

- כיצד זה יראה בקוד שלנו?

$S \rightarrow if\ \boldsymbol{B}$ {emit($2.true || ‘:’);} $then\ \boldsymbol{S_1}$

פתרון עם else

$$S \rightarrow if\ \boldsymbol{B}\ then\ \boldsymbol{M_1 S_1}\ N\ else\ \boldsymbol{M_2 S_2}$$

- נגדיר **מרקר N** שמטרתו דילוג על קטע קוד.
- N בעל תכונה סמנטית next שמטרתה שמירת לייבל שאליו יש לבצע קפיצה בהמשך הקוד.

N → ε { N.next = freshLabel();
emit(‘goto’ || N.next); }

דוגמא

- מה יהיה פלט המנתח שלנו עבור

if 4<5 then f1() else f2()

$$S \rightarrow \; if\ \boldsymbol{B}\ then\ \boldsymbol{M_1}\ \boldsymbol{S_1}\ else\ \boldsymbol{N}\boldsymbol{M_2}\boldsymbol{S_2}$$

$(f1)$ $(f2)$

E_1 or E_1

4 $<$ 5

if 4<5 then f1() else f2()

$S \rightarrow if\ \boldsymbol{B}\ then\ \boldsymbol{M_1}\ \boldsymbol{S_1}\ else\ \boldsymbol{N}\ \boldsymbol{M_2}\ \boldsymbol{S_2}$

פלט

פקודות בבאפר	תכונות
1. t1=4	1. E1.var = t1
2. t2=5	2. E2.var = t2
3. if t1<t2 goto B.trueLabel	3. B prints
4. goto B.falseLabel	4. B prints
5. B.trueLabel: f(1)	5. M prints
6. goto N.next	6. N prints
7. B.falseLabel: f(2)	7. M prints
8. N.next:	8. S prints
9. goto S.next	9. S prints