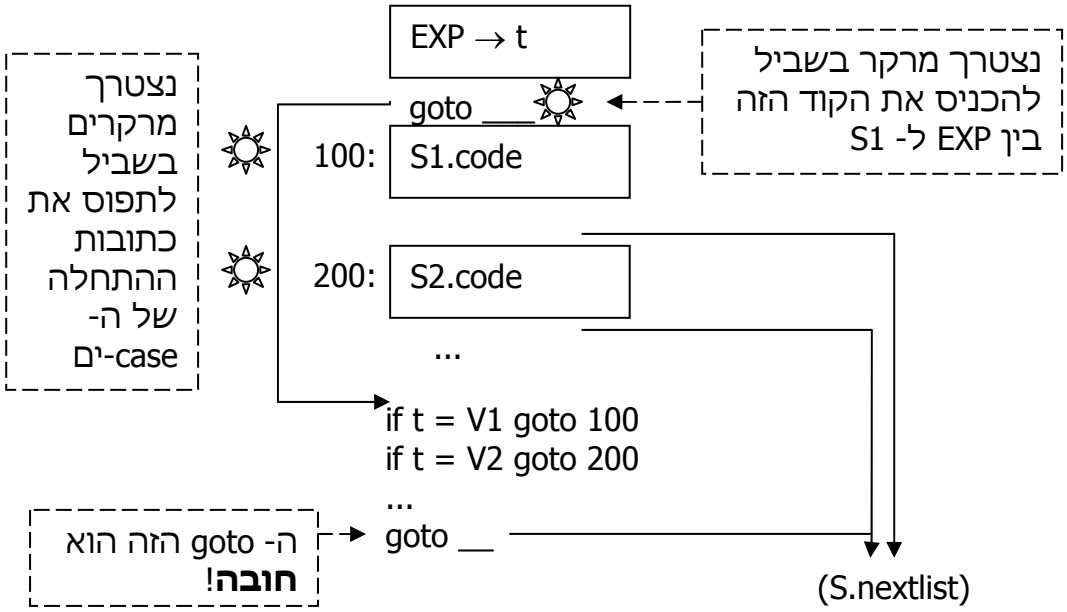
תרגום משפטי switch

הדקדוק:

S	→	switch (EXP) { CASE_LIST }
CASE_LIST	→	CASE CASE_LIST
CASE_LIST	→	CASE
CASE	→	case VALUE : S ; break ;
VALUE	→	const

דיאגרמה:

(תזכורת – השלד של הדיאגרמה הוא הבלוקים של הקוד שנוצרים ע"י משתנים באגף ימין, וביניהם מכניסים קוד שיגרום לקטע כולו לבצע את הנדרש. בזמן שמציירים את הדיאגרמה כדאי לזהות מקומות בהם יהיה צורך להוסיף מרקרים.)



- לא צריך goto-ים באמצע, כי כל S אחראי לקפוץ בעצמו לכתובת הנכונה – רק צריך לעשות backpatch ל- nextlist שלהם.
- ה- goto האחרון הוא **חובה** במקרה זה: אם אף אחד מה- case-ים לא מתקיים, יש לקפוץ לכתובת הבאה לביצוע, שאינה בהכרח השורה הבאה בקוד ביניים (למשל – אם ה- switch כולו נמצא בתוך if, אז אחרי שמסיימים לבצע אותו צריך לקפוץ לסוף ה- if).

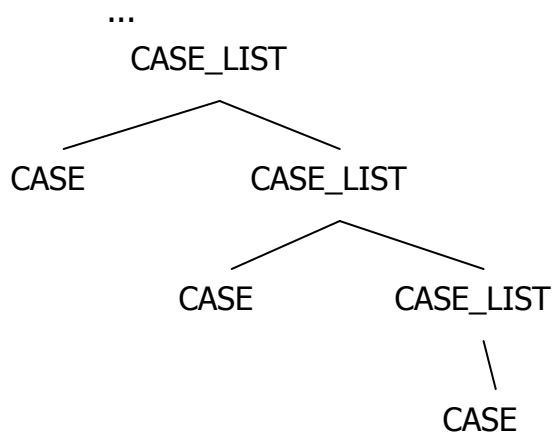
תכונות:

- nextlist ל- CASE, CASE_LIST, S
- quad ל- CASE (כדי שנדע איפה הוא מתחיל)
- val ל- CASE, VALUE
- ל- CASE_LIST: val_list, quad_list

פונקציות:

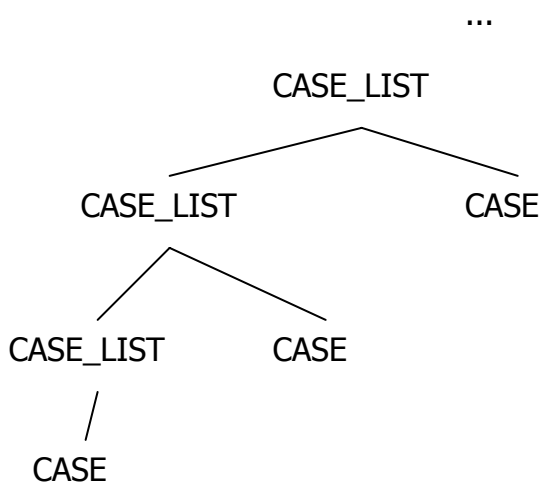
- newStack, pop, push

<u>השפעת כיוון הרקורסיה:</u>
כיוון הרקורסיה (ימנית או שמאלית) קובע את הסדר בו "נפגוש" את ה- case-ים במהלך הניתוח. במקרה שלנו (רקורסיה ימנית) עץ הגזירה יראה כך:



במהלך הניתוח, המנתח מבצע reduce ברגע שאפשר: במקרה שלנו, אי אפשר לבצע reduce ל- CASE_LIST עד שקראנו את כל ה- case-ים בקלט (מומלץ לבנות מנתח SLR עבור שני הכללים של הרקורסיה ולראות למה; לצורך פשטות, אפשר להתייחס ל- CASE כאל טרמינל), ואז ה- reduce הראשון שיתבצע יהיה לפי הכלל CASE_LIST $\rightarrow$ CASE (תנאי העצירה של הרקורסיה) – זה יהיה ה- case הימני ביותר בעץ, שהוא ה**אחרון** בקוד. אם כן, ה- case הראשון שנפגוש במהלך הניתוח הוא האחרון בקוד, ושאר ה- case-ים יתווספו בזה אחר זה בסדר הפוך לסדר הופעתם בקוד. לכן במקרה זה יש לשמור את רשימות התכונות של ה- case-ים (quad_list ו- val_list) כמחסניות, כדי שבסוף נוציא אותן בסדר הנכון.

עבור רקורסיה שמאלית, עץ הגזירה שמתקבל היה נראה כך:



כאן, ברגע שנקרא ה- case הראשון מתבצע reduce ל- CASE_LIST, ואח"כ פוגשים את שאר ה- case-ים לפי סדר הופעתם בקוד. במקרה זה את רשימות התכונות יש לשמור כתור.

המלצה: בנו מנתח SLR עבור הדקדוק ל- switch, והריצו אותו על קלט לדוגמא. וודאו שהדברים מתבצעים בסדר הנכון.

<u>סכימת התרגום</u>
(תזכורת: אחרי שכותבים כלל סמנטי כדאי לעבור על רשימת התכונות שהכנתם קודם, ולוודא שטיפלתם בכל התכונות של המשתנים באגף ימין של הכלל ושייצרתם את כל התכונות של המשתנה באגף שמאל.)

```
VALUE       →   const
    {
            VALUE.val = const.val;
    }

CASE        →   case VALUE : Q S ; break ;
    {
            CASE.nextlist = S.nextlist;
            CASE.quad = Q.quad;
            CASE.val = VALUE.val;
    }

Q           →   ε
    {
            Q.quad = nextquad();
    }

CASE_LIST   →   CASE
    {
            CASE_LIST.nextlist = CASE.nextlist;
            CASE_LIST.count = 1;
            CASE_LIST.val_list = newStack();
            push(CASE_LIST.val_list, CASE.val);
            CASE_LIST.quad_list = newStack();
            push(CASE_LIST.quad_list, CASE.quad);
    }

CASE_LIST   →   CASE  CASE_LIST1
    {
            CASE_LIST.nextlist = merge(CASE_LIST1.nextlist,
CASE.nextlist);
            CASE_LIST.count = CASE_LIST1.count + 1;
            CASE_LIST.val_list = CASE_LIST1.val_list;
            push(CASE_LIST.val_list, CASE.val);
            CASE_LIST.quad_list = CASE_LIST1.quad_list;
            push(CASE_LIST.quad_list, CASE.quad);
    }

S           →   switch ( EXP ) G { CASE_LIST }
```

```
        {
                backpatch(G.nextlist, nextquad());

                // Create the ifs
                for(i = 1; i <= CASE_LIST.count; i++)
                        emit('if ' || EXP.place || ' = ' || pop(CASE_LIST.val_list) ||
'goto' || pop(CASE_LIST.quad_list);

                S.nextlist = merge(CASE_LIST.nextlist, makelist(nextquad()));
                emit("goto __");

        }

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