סיכום שפת ביניים + Backpatch

28 ביולי 2008

שפת הביניים

נשתמש בפקודות הבאות:

```
t_i = val aop val
t_i = val
if(val rop val)goto address
goto address
```

הערות:

- val - משתנה זמני או ערך מספרי (למשל t_9 או 11)
- aop - פעולה אריטמטית (+,-,*,/)
- rop - פעולת השוואה (==,!=,<,>)
- $address$ - כתובת של פקודה בשפת ביניים (מספר או תוית)
- אפשר להרחיב את שפת הביניים על ידי הוספת פקודות קלט פלט.

הדקדוק הבסיסי

S→ if B then S_1
| if B then S_1else S_2

E→ num
| id
| E_1 aop E_2

B→ true
| false
| E rop E
| B or B
| B and B

כאשר המשתנה S גוזר פקודות, E גוזר ביטויים אריטמטים ו B גוזר ביטויים בולאניים.
הערות:

- בנוסף לכללים שמוצגים כאן קיימים בדקדוק כללים נוספים.
- aop- פעולה אריטמטית (+, -, *, /)
- rop - פעולת השוואה (==, !=, <, >, <=, >=)

מרקרים

נשתמש בשני מרקרים:

M→ ϵ

N→ ϵ

תכונות

S:

- list<address> : nextList

E:

- tmp : place

B:

- list<address> : falseList
- list<address> : trueList

M:

- address : quad

N:

- address : nextList

הערות:

- $address$ - כתובת של פקודה בשפת ביניים (מספר או תוית)
- tmp - כתובת של משתנה זמני (s[100] למשל או t_8)
- הרשימות מכילות כתובות של פקודות קפיצה שהודפסו כבר לחוצץ ללא כתובת הקפיצה. תהליך ה $backpatching$ יעדכן את כתובת הקפיצהה במקומות החסרים.

פונקציות עזר

```
void emit(string code)
```

מדפיסה את הקוד לחוצץ.

```
address nextQuad()
```

מחזירה את הכתובת הבאה בחוצץ (הכתובת אליה תודפס הפקודה הבאה)

```
tmp newTmp()
```

מחזירה משתנה זמני חדש (אליו אפשר להציב ערך)

```
void backpatch(list<address> list, address dest)
```

מטליאה את הפקודות ברשימה עם כתובת היעד $dest$.

```
list<address> makeList(address a)
```

מייצרת רשימה חדשה שמכילה את הכתובת a.

```
list<address> merge(list<address> l1, list<address> l2)
```

מחזירה רשימה חדשה שמכילה את כל הכתובות שהיו ב $l1$ ו $l2$.

כללים סמנטים

E→ num

```
E.place = newTmp()
emit(E.place = num.value)
```

E→ id

```
E.place = newTmp()
emit(E.place = symbolTable.getPlace(id.name))
```

E→ E_1aop E_2

```
E.place = newTmp()
emit(E.place = E1.place aop E2.place)
```

B→ true

```
B.trueList = makeList(nextQuad())
emit(goto__)
```

B→ | false

```
B.falseList = makeList(nextQuad())
emit(goto__)
```

B→ | E_1 rop E_2

```
B.trueList = makeList(nextQuad())
emit(if(E1.place rop E2.place)goto__)
B.falseList = makeList(nextQuad())
emit(goto__)
```

B→ B_1 or M B_2

```
backpatch(B1.falseList, M.quad)
B.falseList = B2.falseList
B.trueList = merge(B1.trueList, B2.trueList)
```

B→ B_1 and M B_2

```
backpatch(B1.trueList, M.quad)
B.trueList = B2.trueList
B.falseList = merge(B1.falseList, B2.falseList)
```

S→ if B then M S_1

```
backpatch(B.trueList, M.quad)
S.nextList = merge(B.falseList, S1.nextList)
```

S→ if B then M_1 S_1else N M_2 S_2

```
backpatch(B.trueList, M1.quad)
backpatch(B.falseList, M2.quad)
S.nextList = merge(merge(S1.nextList, S2.nextList), makeList(N.quad))
```

M→ ϵ

```
M.quad = nextQuad()
```

N→ ϵ

```
N.quad = nextQuad()
emit(goto__)
```