四、中间代码生成
(2. 表达式的翻译)

魏恒峰

hfwei@nju.edu.cn

2023 年 05 月 06 日

产生式	语义规则
$\boxed{S \rightarrow \textbf{id} = E\ ;}$	$S.code = E.code \parallel$ $gen(top.get(\textbf{id}.lexeme)\ '='\ E.addr)$
$\boxed{E} \rightarrow E_1 + E_2$	$E.addr = \textbf{new}\ Temp\,()$ $E.code = E_1.code \parallel E_2.code \parallel$ $gen(E.addr\ '='\ E_1.addr\ '+'\ E_2.addr)$
$\mid\ -E_1$	$E.addr = \textbf{new}\ Temp\,()$ $E.code = E_1.code \parallel$ $gen(E.addr\ '='\ '\textbf{minus}'\ E_1.addr)$
$\mid\ (\ E_1\ )$	$E.addr = E_1.addr$ $E.code = E_1.code$
$\mid\ \textbf{id}$	$E.addr = top.get(\textbf{id}.lexeme)$ $E.code = ''$

$\boxed{S \rightarrow \textbf{id} = E}\ ;$	$\{\ gen(\ top.get(\textbf{id}.lexeme)\ '='\ E.addr);\ \}$
$\mid\ \boxed{L = E}\ ;$	$\{\ gen(L.array\ .base\ '['\ L.addr\ ']'\ '='\ E.addr);\ \}$
$E \rightarrow E_1 + E_2$	$\{\ E.addr = \textbf{new}\ Temp\,();$ $gen(E.addr\ '='\ E_1.addr\ '+'\ E_2.addr);\ \}$
$\mid\ \textbf{id}$	$\{\ E.addr = top.get(\textbf{id}.lexeme);\ \}$
$\mid\ \boxed{L}$	$\{\ E.addr = \textbf{new}\ Temp\,();$ $gen(E.addr\ '='\ L.array.base\ '['\ L.addr\ ']');\ \}$
$L \rightarrow \textbf{id}\ [\ E\ ]$	$\{\ L.array = top.get(\textbf{id}.lexeme);$ $L.type = L.array.type.elem;$ $L.addr = \textbf{new}\ Temp\,();$ $gen(L.addr\ '='\ E.addr\ '*'\ L.type.width);\ \}$
$\mid\ L_1\ [\ E\ ]$	$\{\ L.array = L_1.array;$ $L.type = L_1.type.elem;$ $t = \textbf{new}\ Temp\,();$ $L.addr = \textbf{new}\ Temp\,();$ $gen(t\ '='\ E.addr\ '*'\ L.type.width);$ $gen(L.addr\ '='\ L_1.addr\ '+'\ t);\ \}$

表达式的中间代码翻译

综合属性 $E.code$: 中间代码

产生式	语义规则
$\boxed{S \rightarrow \mathbf{id} = E\ ;}$	$S.code = E.code \parallel gen(top.get(\mathbf{id}.lexeme)\ '='\ E.addr)$
$\boxed{E} \rightarrow E_1 + E_2$	$E.addr = \mathbf{new}\ Temp()$ $E.code = E_1.code \parallel E_2.code \parallel gen(E.addr\ '='\ E_1.addr\ '+'\ E_2.addr)$
$\mid\ -E_1$	$E.addr = \mathbf{new}\ Temp()$ $E.code = E_1.code \parallel gen(E.addr\ '='\ '\mathbf{minus}'\ E_1.addr)$
$\mid\ (E_1)$	$E.addr = E_1.addr$ $E.code = E_1.code$
$\mid\ \mathbf{id}$	$E.addr = top.get(\mathbf{id}.lexeme)$ $E.code = ''$

$$a = b + -c$$

```
t1 = minus c
t2 = b + t1
a = t2
```

综合属性 $E.addr$: 变量名 (包括临时变量)、常量

```c
int main() {
  int a = 0, b = 1, c = 2;

  a = b + -c;

  return 0;
}
```

```
%7 = sub nsw i32 0, %6
%8 = add nsw i32 %5, %7
store i32 %8, i32* %2, align 4
```

A wonderful tool: Compiler Explorer

数组引用的中间代码翻译

声明 : $\text{int } a[2][3]$

数组引用 : $x = a[1][2];\ a[1][2] = x$

需要计算 $a[1][2]$ 相对于**数组基地址** a 的**偏移地址**

$$addr(a[1][2]) = base + 1 \times 12 + 2 \times 4$$

	类型	宽度
a	$array(2, array(3, \text{integer}))$	24
$a[i]$	$array(3, \text{integer})$	12
$a[i][j]$	integer	4

int $a[2][3]$

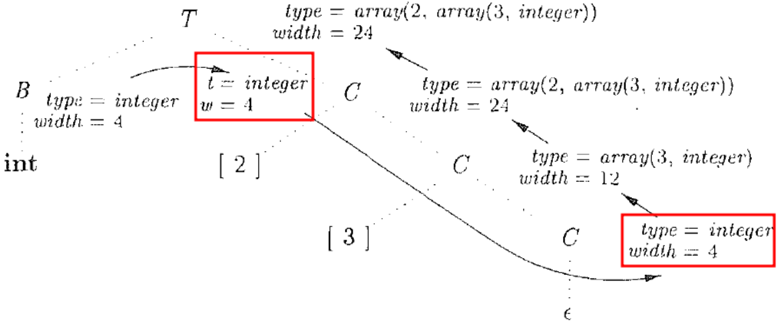
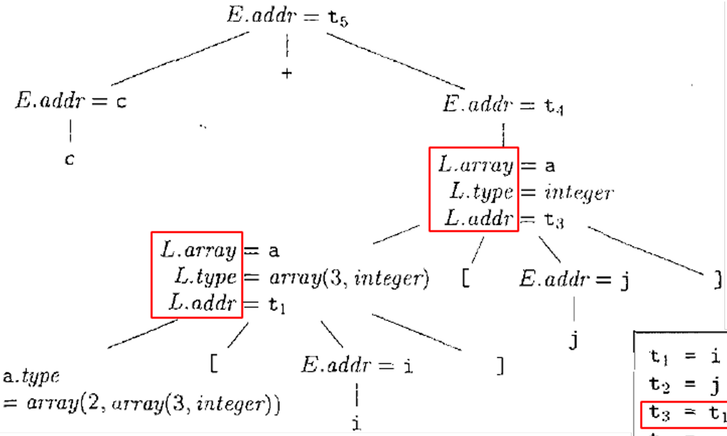


图 6-16　数组类型的语法制导翻译

综合属性 $L.array(.base)$：数组基地址（即，数组名）

S	$\rightarrow$	$\textbf{id} = E\ ;$	$\{\ gen(\ top.get(\textbf{id}.lexeme)\ '='\ E.addr);\ \}$
	$\mid$	$\boxed{L = E\ ;}$	$\{\ gen(\boxed{L.array}.base\ '['\ \boxed{L.addr}\ ']'\ '='\ E.addr);\ \}$
E	$\rightarrow$	$E_1 + E_2$	$\{\ E.addr = \textbf{new}\ Temp\,();$ $gen(E.addr\ '='\ E_1.addr\ '+'\ E_2.addr);\ \}$
	$\mid$	$\textbf{id}$	$\{\ E.addr = top.get(\textbf{id}.lexeme);\ \}$
	$\mid$	$\boxed{L}$	$\{\ E.addr = \textbf{new}\ Temp\,();$ $gen(E.addr\ '='\ \boxed{L.array}.base\ '['\ \boxed{L.addr}\ ']');\ \}$
L	$\rightarrow$	$\boxed{\textbf{id}\ [\ E\ ]}$	$\{\ \boxed{L.array} = top.get(\textbf{id}.lexeme);$ $L.type = L.array.type.elem;$ $L.addr = \textbf{new}\ Temp\,();$ $gen(L.addr\ '='\ E.addr\ \boxed{'*'}\ L.type.width);\ \}$
	$\mid$	$\boxed{L_1\ [\ E\ ]}$	$\{\ \boxed{L.array} = L_1.array;$ $L.type = L_1.type.elem;$ $t = \textbf{new}\ Temp\,();$ $L.addr = \textbf{new}\ Temp\,();$ $gen(t\ '='\ E.addr\ '*'\ L.type.width);$ $gen(L.addr\ '='\ L_1.addr\ \boxed{'+'}\ t);\ \}$

综合属性 $L.addr$：偏移地址




```
t1 = i * 12
t2 = j * 4
t3 = t1 + t2
t4 = a [ t3 ]
t5 = c + t4
```

$$c + a[i][j]$$

```
%2 = alloca [2 x [3 x i32]], align 16
```

```c
int main() {
  int a[2][3] = { 0 };

  int i = 1, j = 2;
  int c = 10, d = 20;

  d = c + a[i][j];

  return 0;
}
```

```
%8 = load i32, i32* %5, align 4  %8: c
%9 = load i32, i32* %3, align 4  %9: i
%10 = sext i32 %9 to i64
%11 = getelementptr inbounds [2 x [3 x i32]], [2 x [3 x i32]]* %2, i64 0, i64 %10
%12 = load i32, i32* %4, align 4  %12: j
%13 = sext i32 %12 to i64
%14 = getelementptr inbounds [3 x i32], [3 x i32]* %11, i64 0, i64 %13
%15 = load i32, i32* %14, align 4  %15: a[i][j]
%16 = add nsw i32 %8, %15
store i32 %16, i32* %6, align 4
```

GEP provides a way to **access arrays and manipulate pointers**.

GEP abstract away details like **size of types**.

```
getelementptr
<base-type>, <base-type>* <base-addr>, [i32 <index>]+
```

`<base-type>`: base type used for **the first** index

- It does *not* change the pointer type.
- It offsets by the `<base-type>`.

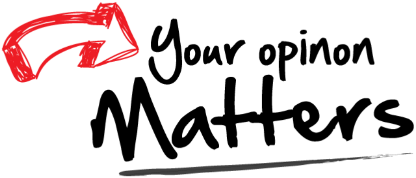
Further indices:

- Offset inside arrays (aggregate types)
- *Change* the pointer type by removing one layer of “aggregation”

The Often Misunderstood GEP Instruction @ LLVM Docs

LLVM IR Tutorial: Phis, GEPs and Other Things, Oh My! @ bilibili


Thank
You!



Office 926

hfwei@nju.edu.cn