Lec18 CS241

Graham Cooper

July 8th, 2015

Grammar Rules

Expressions

Address of

$$\frac{E{:}int}{\&E{:}int*}$$

If E has type int, then &E has type int *

Dereference:

$$\frac{E{:}int*}{*E{:}int}$$

IF E has type int * then *E has type int

Allocation:

$$\frac{E{:}int}{newint[E]{:}int*}$$

Mult:

$$\frac{E_1{:}intE_2{:}int}{E_1*E_2{:}int}$$

Similarily for /, %

Addition:

$$\frac{E_1{:}int \cap E_2{:}int}{E_1+E_2{:}int}$$

$$\frac{E_1{:}int* \cap E_2{:}int}{E_1+E_2{:}int*}$$

$$\frac{E_1{:}int \cap E_2{:}int*}{E_1+E_2{:}int*}$$

Substraction:

$$\frac{E_1{:}int \cap E_2{:}int}{E_1+E_2{:}int}$$

$$\frac{E_1{:}int* \cap E_2{:}int}{E_1+E_2{:}int*}$$

$$\frac{E_1{:}int* \cap E_2{:}int*}{E_1+E_2{:}int}$$

Functions:

$$\frac{<f,(\tau_1,\tau_2...\tau_n)> \in procs E_1{:}\tau_1...E_n{:}\tau_n}{F(E_1,...E_n){:}int}$$

Additional Type Constrains

Loops, if statements
while(T) {S}
if (T) {S_1} else {S_2}

Test T hsould be boolean, not int or int*. But there is not boolean type in WLP4
The grammar ensures that T will be a boolean tests (Expr comparison Expr), so as long as the comparison is well-typed, the if/while will be correct

Any expression that has a type is well-typed.
$\frac{E{:}\tau}{well{-}typed(E)}$

Tests:
$\frac{E_1{:}\tau \cap E_2{:}\tau}{well{-}typed(E_1 == E_2)}$
similar with !=

$\frac{E_1{:}\tau \cap E_2{:}\tau}{well{-}typed(E_1 < E_2)}$
Similar with >, >=, <=

Assignment:
$\frac{E_1{:}\tau \cap E_2{:}\tau}{well{-}typed(E_1 = E_2)}$

Print:
$\frac{E{:}int}{well{-}typed(println(E))}$

Deallocation:
$\frac{E{:}int*}{well{-}typed(delete[]E)}$

IF:
$\frac{well{-}typed(T) \cap well{-}typed(S_1) \cap well{-}typed(S_2)}{well{-}typed(if(T)(\{S_1\}else\{S_2\})}$

Loops:
$\frac{well{-}typed(T) \cap well{-}typed(S)}{well{-}typed(while(T)\{S\})}$

Sequences:

$\frac{}{well{-}typed(\epsilon)}$

$$\frac{well-typed(S_1) \cap well-typed(S_2)}{well-typed(S_1,S_2)}$$

Dcls:

$$\frac{}{well-typed(intid=NUM)}$$

$$\frac{}{well-typed(int*id=NULL)}$$

Procedures:

$$\frac{well-typed(dcls) \cap well-typed(stmts) \cap E{:}int}{well-typed(INT,ID,(params)\{dcls,stmts,return,E;\})}$$

WAIN:

$$\frac{dcl_2{=}INT,ID \cap well-typed(dcls) \cap well-typed(stmts) \cap E{:}int}{well-typed(INT,WAIN(dcl,dcl_2)\{dcls,stmts,return,E;\})}$$

Code Generation

Source $\rightarrow$ Lexical Analysis $\overset{tokens}{\rightarrow}$ parsing $\rightarrow$ semantic analysys $\overset{parsetreeandsymboltable}{\rightarrow}$ Code Generation $\rightarrow$ assembly

- By now, the source pgoram is guaranteed to be free of compile-time errors.
- We must now output equivalent MIPS code

There are ∞ many equivalent MIPS programs, which do we choose?

- correct (essential)
- easiest (for cs241)
- shortest
- fastest to run
- fastest to compile

0.1 Example:

```
int wain(int a, int b) { return a;}
```

Convention:
- parameters of wain are in $1 and $2
- output will be in $3

MIPS:
add $3, $1, $0
jr $31

```
int wain (int a, int b) {return b;}
```

MIPS:
add $3, $2, $0
jr $31

Parse trees for these programs are the same.
How do we tell one from the other?
Determine which of the two declared ID's the returned ID matches

How do we do that? Use the symbol table.
Add a field to the symbol table to indicate where each symbol is stored

Name	TYpe	Location
a	int	$1
b	int	$2

Then when traversing for code gen, look up ID in the symbol table
a $\rightarrow$ $1
b $\rightarrow$ $2

Local Declerations?
- cant all be in regs, probably not enough
For simplicity:

- put all local vars on the stack, including params of wain.

```
int wain(int a, int b){return a;}
```

```
sw $1, -4 ($30)
sw $2, -8($30)
```

```
lis $4
.word 8
sub $30, $30, $4
lw $3, 4($30)
add $30, $30, $4
jr $31
```