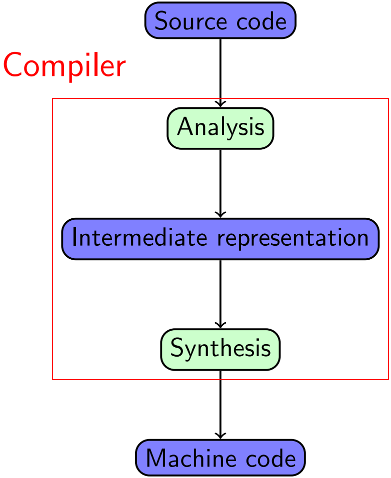
Compiler Construction

Chapter 5 – Code Generation (1)

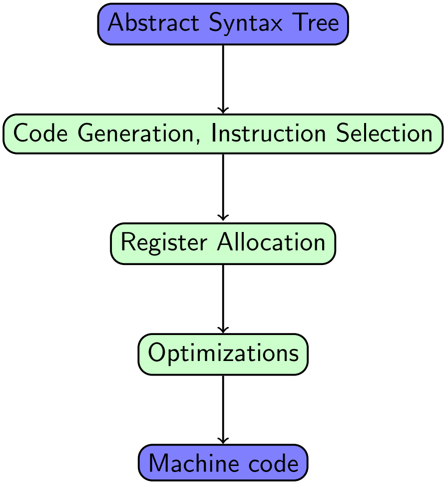
Prof. Dr. Jörg Kreiker
joerg.kreiker@informatik.hs-fulda.de

Fachbereich Angewandte Informatik
Hochschule Fulda – University of Applied Sciences

Compiler Overview



Synthesis Phase



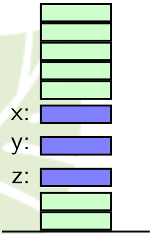
Agenda

Hochschule Fulda
University of Applied Sciences

Code generation for

1. Expressions
2. (Sequence of) Statements
3. Conditionals
4. Loops
5. Arrays
6. Records
7. Pointer
8. Functions
9. Whole programs

Variables are stored in stack S at a certain address



- Associating variables with addresses happens during symbol table creation
- Addresses are stored in declaration nodes in the AST
- Every usage of a variable has a pointer to the variable's declaration node
- In the following, we assume an address environment ρ, which yields the (relative) address of each variable

L-Value and R-Value

- Variables are used in two different ways
- In an assignment `x = y + 1`, we are interested in the content of `y` and in the address of `x`
- l-value of `x`: address
- r-value of `x`: content

$\text{code}_R\ e\ \rho$	generates code to compute the r-value of e in address environment ρ
$\text{code}_L\ e\ \rho$	analogous for l-value

$\text{code}_R\ e_1 + e_2\ \rho$	$=$	$\text{code}_R\ e_1\ \rho$ $\text{code}_R\ e_2\ \rho$ `add` analogous for `mul`, ...
$\text{code}_R\ (-e)\ \rho$	$=$	$\text{code}_R\ e_1\ \rho$ `neg` analogous for `not`, ...
$\text{code}_R\ q\ \rho$	$=$	`loadc q`
$\text{code}_L\ x\ \rho$	$=$	`loadc` $\rho(x)$
$\text{code}_R\ x\ \rho$	$=$	$\text{code}_L\ x\ \rho$ `load`
$\text{code}_R\ x = e\ \rho$	$=$	$\text{code}_R\ e\ \rho$ $\text{code}_L\ x\ \rho$ `store`

- If e is an expression, then e; is a statement
- Statements do not return a value, hence the stack pointer is the same before and after
- code $e;\ \rho$ generates such code
- Note below: *stm* is a statement *stms* is a sequence of statements

code $e;\ \rho$	$=$	$\text{code}_R\ e\ \rho$ pop
code $stm\ stms\ \rho$	$=$	code $stm\ \rho$ code $stms\ \rho$

Code Generation for If

- Let s be the statement `if` (c) *stms*
- Generate code to evaluate c and to execute *stms*
- In between jump on zero to behind the code generated for `stms`

$$
\begin{array}{lcl}
\text{code } s\ \rho & = & \text{code}_R\ c\ \rho \\
 & & \texttt{jumpz A} \\
 & & \text{code } stms\ \rho \\
 & \text{A:} & \ldots
\end{array}
$$

Code Generation for If-Else

- Let s be the statement `if (c)` tt `else` ee
- Use similar strategy

$$
\boxed{
\begin{array}{lll}
\text{code } s\ \rho & = & \text{code}_R\ c\ \rho \\
 & & \texttt{jumpz A} \\
 & & \text{code } tt\ \rho \\
 & & \texttt{jump B} \\
 & \text{A:} & \text{code } ee\ \rho \\
 & \text{B:} & \ldots
\end{array}
}
$$

Example

Let $\rho = \{x \mapsto 4, y \mapsto 7\}$ and generate code s ρ for s being

$\texttt{if } (x > y) \qquad x = x - y; \qquad \texttt{else } y = y - x;$

- Let s be the statement `while` (e) s'
- We generate code as follows:

$$
\begin{array}{lll}
\text{code } s\ \rho & = & \\
 & \text{A:} & \text{code}_R\ e\ \rho \\
 & & \texttt{jumpz B} \\
 & & \text{code } s'\ \rho \\
 & & \texttt{jump A} \\
 & \text{B:} & \ldots
\end{array}
$$

Code Generation for For

- Let s be the statement `for` $(e_1; e_2; e_3)\ s'$
- Equivalent to $e_1;$ `while` $(e_2)\ \{s'\ e_3;\}$
- Therefore ...

code $s\ \rho$	$=$	$\text{code}_R\ e_1\ \rho$
		`pop`
	A:	$\text{code}_R\ e_2\ \rho$
		`jumpz B`
		code $s'\ \rho$
		$\text{code}_R\ e_3\ \rho$
		`pop`
		`jump A`
	B:	...

Example

Let $\rho = \{a \mapsto 7, b \mapsto 8\; c \mapsto 9\}$ and generate code $s\ \rho$ for s being

$$\texttt{while}\ (a > 0)\ \{c = c + 1;\ \ a = a - b;\}$$

Arrays

- Consider `int[11] a;`
- Array `a` contains 11 elements hence it occupies 11 cells
- $\rho(a)$ is the address of element `a[0]`

Size of Types

- We need to define the size $|\cdot|$ of a type (as in `sizeof`)

$$|t| = \begin{cases} 1 & \text{for base types } t \\ k \cdot |t'| & \text{if } t \equiv t'[k] \end{cases}$$

- For declarations $d \equiv t_1\ x_1; \ldots t_k\ x_k;$

$$\begin{array}{rcll} \rho(x_1) & = & 0 & \\ \rho(x_i) & = & \rho(x_{i-1}) + |t_{i-1}| & \quad \text{for } i > 1 \end{array}$$

- ρ and $|\cdot|$ are computed at compile time

- In C, an array is a pointer. A declared array a is a pointer constant with r-value the start address of a
- Hence for array e: $\text{code}_R\ e\ \rho = \text{code}_L\ e\ \rho$
- Let `t[c] a;` be an array declaration

$$\boxed{\begin{array}{lcl} \text{code}_L\ e_1[e_2]\ \rho & = & \text{code}_R\ e_1\ \rho \\ & & \text{code}_R\ e_2\ \rho \\ & & \texttt{loadc}\ |t| \\ & & \texttt{mul} \\ & & \texttt{add} \end{array}}$$

- Let `struct { int a; int b;} x;` be part of a declaration list
- `x` gets the first free cell of the record's space as relative address
- The components get addresses relative to the beginning of the record
  - $\rightarrow$ Here $\{a \mapsto 0, b \mapsto 1\}$
- Size and addresses of record t with component types t_i

$$|t| = \Sigma_{i=1}^{k} |t_i| \qquad \rho(c_1) = 0 \qquad \rho(c_i) = \rho(c_{i-1}) + |t_{i-1}|$$

$$\boxed{\begin{array}{lcl} \text{code}_L\ e.c\ \rho & = & \text{code}_L\ e\ \rho \\ & & \texttt{loadc}\ \rho(c) \\ & & \texttt{add} \end{array}}$$