Phases of a compiler

Intermediate Representation (IR): specification and generation

Figure 1.6, page 5 of text
Issues to consider

- Array access
  - Example 6.25 & bounds checking
Function calls

- Basic form: \text{id}(e_1,e_2,\ldots,e_k)

- General form: \text{assignable}(e_1,e_2,\ldots,e_k)

- If \( f \) is a function, \( g(4,5) \) returns a function, and \( r.h \) yields a function, then the following are legal:

\[
\begin{align*}
  f(3) & \quad g(4,5)(3) & \quad r.h(3)
\end{align*}
\]
How is function call carried out?

1. evaluate each of the argument expressions

2. mark the resulting values as parameters

3. invoke the function
examples

f(x+1)
examples

\[ f(x+1) \]

Remember that the function call has structure.
examples

\[ f(x+1) \]

\[ t1 = x + 1 \]

Generate code for the argument expression
examples

\[ f(x+1) \]

\[ t1 = x + 1 \]

param \ t1

Mark the result as a parameter of the function call
Call the function. The second argument of the call indicates the arity of the function (i.e. how many parameters it has).
examples

\[ f(x+1) \quad f(x+1,2\cdot y) \]

\[ t1 = x + 1 \]
\[ \text{param } t1 \]
\[ \text{call}(f,1) \]
examples

\[ f(x+1) \quad f(x+1, 2y) \]

\[ t1 = x + 1 \]
\[ \text{param } t1 \]
\[ \text{call}(f, 1) \]

Remember that the function call has structure.
examples

\[ f(x+1) \quad f(x+1,2*y) \]

\[ t1 = x + 1 \quad t1 = x + 1 \]

param t1
call(f,1)

Evaluate the first argument expression.
examples

\[ f(x+1) \]

\[ t1 = x + 1 \]
\[ \text{param } t1 \]
\[ \text{call}(f,1) \]

\[ f(x+1,2*\text{y}) \]

\[ t1 = x + 1 \]
\[ \text{param } t1 \]

Mark the result as a parameter.
The examples shown demonstrate the use of parameters and functions in a program. The first example introduces a parameter `t1` set to `x + 1`, and then calls a function `f` with `t1` as an argument. The second example is similar but with an additional parameter `t2` set to `2 * y`, and both parameters are used in the function call. The comment suggests evaluating the second argument expression, indicating a focus on understanding and manipulating expressions within a program.
examples

\[ f(x+1) \]

\[ t_1 = x + 1 \]
\[ \text{param } t_1 \]
\[ \text{call}(f,1) \]

\[ f(x+1,2*y) \]

\[ t_1 = x + 1 \]
\[ \text{param } t_1 \]
\[ t_2 = 2 \times y \]
\[ \text{param } t_2 \]

Mark the result as a parameter.
examples

\[
f(x+1) \quad \rightarrow \quad f(x+1,2*y)
\]

\[
t1 = x + 1 \\
param t1 \\
call(f,1)
\]

\[
t1 = x + 1 \\
param t1 \\
t2 = 2 * y \\
param t2 \\
call(f,2)
\]

Call the function.
An alternative to intermingling the 'param' instructions with the argument evaluation is to gather them in a queue, then place them between the argument evaluations and before the 'call' instruction.
A slightly more involved example.

\[ f(x+1) \]
\[ f(x+1, 2y) \]
\[ t1 = x + 1 \]
\[ \text{param } t1 \]
\[ \text{call}(f, 1) \]
\[ t2 = 2y \]
\[ \text{param } t2 \]
\[ \text{call}(f, 2) \]
\[ f(g(3z), h(a+b, a\cdot b)) \]
\[ t1 = x + 1 \]
\[ \text{param } t1 \]
\[ \text{call}(f, 1) \]
\[ t2 = 2y \]
\[ \text{param } t2 \]
\[ \text{call}(f, 2) \]
exercise

What intermediate code do you come up with for this example?

\[ f(g(3z), h(a+b, a*b)) \]
As before, remember the structure...
examples

\[ f(x+1) \]
\[ t1 = x + 1 \]
\[ \text{param } t1 \]
\[ \text{call}(f,1) \]

\[ f(x+1,2*y) \]
\[ t1 = x + 1 \]
\[ t2 = 2 * y \]
\[ \text{param } t1 \]
\[ \text{param } t2 \]
\[ \text{call}(f,2) \]

\[ f(g(3*z),h(a+b,a*b)) \]
\[ t1 = 3 * z \]
\[ \text{param } t1 \]
\[ \text{param } t2 \]
\[ \text{call}(f,2) \]

...but not just the top-level structure!
This translation will happen automatically due to the recursive structure of the function call for f...
examples

\[ f(x+1) \]  \[ f(x+1,2*y) \]  \[ g(3*z) \]

\[ t_1 = x + 1 \]  \[ t_1 = x + 1 \]  \[ t_1 = 3 * z \]
\[ \text{param } t_1 \]  \[ \text{param } t_1 \]  \[ \text{param } t_1 \]
\[ t_2 = 2 * y \]  \[ t_2 = 2 * y \]  \[ \text{param } t_2 \]
\[ \text{param } t_2 \]  \[ \text{param } t_2 \]
\[ \text{call}(f,1) \]  \[ \text{call}(f,1) \]  \[ \text{call}(g,1) \]
\[ \text{call}(f,2) \]  \[ \text{call}(f,2) \]  \[ \text{call}(f,2) \]

...view this as a function call in isolation.
examples

\[
f(x+1)
\]
\[
t_1 = x + 1
\]
\[
\text{param } t_1
\]
\[
\text{call}(f,1)
\]

\[
f(x+1,2*y)
\]
\[
t_1 = x + 1
\]
\[
t_2 = 2 \times y
\]
\[
\text{param } t_1
\]
\[
\text{param } t_2
\]
\[
\text{call}(f,2)
\]

\[
f(g(3*z),h(a+b,a*b))
\]
\[
t_1 = 3 \times z
\]
\[
\text{param } t_1
\]
\[
t_2 = \text{call}(g,1)
\]
\[
\text{param } t_2
\]

Mark the result as a parameter.
examples

\[
f(x+1) \quad f(x+1,2*y) \quad f(g(3*z),h(a+b,a*b))
\]

\[
t_1 = x + 1 \quad t_1 = x + 1 \quad t_1 = 3 * z
\]
\[	param t_1 \quad \param t_1 \quad \param t_1
\]
\[
t_2 = 2 * y \quad \param t_2 \quad \call(g,1)
\]
\[	param t_2 \quad \param t_2 \quad \param t_2
\]
\[
call(f,1) \quad call(f,2) \quad call(f,2)
\]

More structure!
examples

\[ f(x+1) \]
\[ t_1 = x + 1 \]
\[ \text{param } t_1 \]
\[ \text{call}(f,1) \]

\[ f(x+1,2*y) \]
\[ t_1 = x + 1 \]
\[ t_2 = 2 * y \]
\[ \text{param } t_1 \]
\[ \text{param } t_2 \]
\[ \text{call}(f,2) \]

\[ f(g(3*z),h(a+b,a*b)) \]
\[ t_1 = 3 * z \]
\[ \text{param } t_1 \]
\[ \text{param } t_2 \]
\[ \text{call}(g,1) \]
\[ \text{call}(f,2) \]
\[ t_3 = a + b \]
examples

\[
f(x + 1) \quad f(x + 1, 2 \cdot y) \quad f(g(3 \cdot z), h(a + b, a \cdot b))
\]

\[
t_1 = x + 1 \quad t_1 = x + 1 \quad t_1 = x + 1
\]
\[
\text{param } t_1 \quad \text{param } t_1 \quad \text{param } t_1
\]
\[
call(f, 1) \quad t_2 = 2 \cdot y \quad \text{param } t_2
\]
\[
\text{param } t_2 \quad \text{param } t_2 \quad \text{param } t_2
\]
\[
call(f, 2) \quad \text{call}(f, 2) \quad \text{call}(f, 2)
\]

parameter marking
examples

\begin{align*}
\text{f}(x+1) & \quad \text{f}(x+1,2*y) & \quad \text{f}(g(3*z),h(a+b,a*b)) \\
t1 = x + 1 & \quad t1 = x + 1 & \quad t1 = x + 1 \\
\text{param } t1 & \quad \text{param } t1 & \quad \text{t1} = \text{call}(g,1) \\
call(f,1) & \quad \text{param } t2 & \quad \text{param } t2 \\
 & \quad \text{call}(f,2) & \quad \text{call}(f,2) \\
t2 = 2 * y & \quad \text{t2} = a + b & \quad \text{t4} = a * b \\
\end{align*}

expression
examples

\[ f(x+1) \]
\[ f(x+1, 2 \cdot y) \]
\[ t_1 = x + 1 \quad t_1 = x + 1 \quad t_1 = x + 1 \]
\[ \text{param } t_1 \quad \text{param } t_1 \quad \text{param } t_1 \]
\[ \text{call}(f, 1) \quad \text{call}(f, 1) \quad \text{call}(f, 1) \]
\[ t_2 = 2 \cdot y \quad t_2 = 2 \cdot y \quad t_2 = 2 \cdot y \]
\[ \text{param } t_2 \quad \text{param } t_2 \quad \text{param } t_2 \]
\[ \text{call}(f, 2) \quad \text{call}(f, 2) \quad \text{call}(f, 2) \]

\[ f(g(3 \cdot z), h(a+b, a \cdot b)) \]
\[ t_1 = 3 \cdot z \]
\[ \text{param } t_1 \]
\[ t_2 = \text{call}(g, 1) \]
\[ \text{param } t_2 \]
\[ t_3 = a + b \]
\[ \text{param } t_3 \]
\[ t_4 = a \cdot b \]
\[ \text{param } t_4 \]
\[ t_5 = \text{call}(h, 2) \]

parameter marking and call
examples

\[ f(x+1) \]
\[ t_1 = x + 1 \]
param t₁
\[ \text{call}(f,1) \]

\[ f(x+1,2*y) \]
\[ t_1 = x + 1 \]
param t₁
\[ t_2 = 2 \times y \]
param t₂
\[ \text{call}(f,2) \]

\[ f(g(3*z),h(a+b,a*b)) \]
\[ t_1 = 3 \times z \]
param t₁
\[ t_2 = \text{call}(g,1) \]
param t₂
\[ t_3 = a + b \]
param t₃
\[ t_4 = a \times b \]
param t₄
\[ t_5 = \text{call}(h,2) \]
param t₅
\[ \text{call}(f,2) \]

parameter marking and call
examples

\[
f(x+1)
\]
\[
t_1 = x + 1
\]
\[
param \ t_1
\]
\[
call(f,1)
\]

\[
f(x+1,2*y)
\]
\[
t_1 = x + 1
\]
\[
t_2 = 2 \times y
\]
\[
param \ t_1
\]
\[
param \ t_2
\]
\[
call(f,2)
\]

\[
f(g(3*z),h(a+b,a*b))
\]
\[
t_1 = 3 \times z
\]
\[
param \ t_1
\]
\[
t_2 = call(g,1)
\]
\[
param \ t_2
\]
\[
t_3 = a + b
\]
\[
param \ t_3
\]
\[
t_4 = a \times b
\]
\[
param \ t_4
\]
\[
t_5 = call(h,2)
\]
\[
param \ t_5
\]
\[
call(f,2)
\]

Alternate translation gathering 'param' instructions together with call to function.
exercise

- How will you modify your grammar rules to generate intermediate code for function calls?

- Assume that type checking and argument list length checking has already been accounted for in the semantic actions attached to productions:
  - type checking of each argument with corresponding parameter declaration (remembering that there is no coercion allowed in either an explicit or an implicit assignment)
  - checking that the number of arguments and the number of parameters is the same
exercise outcome

- Basic approach teams took was to gather up information about argument expressions in an expression list, and generate the 'param' instructions at the end of the 'assignable ablock' rule, but only if assignable is a function (as opposed to an array). After the param instructions have been generated the 'call' instruction is generated, including the arity of the function (which is determined either by looking it up in the symbol table or by counting the number of arguments supplied).