1 Type

We will summarize some of the features of the following different evaluation strategies that we have in \texttt{ImpNameSmallA}: call-by-value (both \texttt{val} and \texttt{var} annotations), call-by-reference (\texttt{ref}) and call-by-name (\texttt{name}). We will focus on the effects these evaluation strategies have on memory, side effects and at what step exactly memory operations occur.

We will do so by looking at the following simple example:

\begin{verbatim}
var x = 5 in
((pann y: t) => y = y+1; y)(print(7); x);
print(x)
\end{verbatim}

This program captures the following 4 cases:

- call-by-value(\texttt{val})
\begin{verbatim}
var x = 5 in
((y: t) => y+1; y)(print(7); x);
print(x)
\end{verbatim}

- call-by-value(\texttt{var})
\begin{verbatim}
var x = 5 in
((var y: t) => y = y+1; y)(print(7); x);
print(x)
\end{verbatim}

- call-by-reference(\texttt{ref})
\begin{verbatim}
var x = 5 in
((ref y: t) => y+1; y)(print(7); x);
print(x)
\end{verbatim}
• call-by-name(name)

```latex
\text{var } x = 5 \text{ in }
\begin{align*}
((\text{name } y: t) & \Rightarrow y+1; y)(\text{print}(7); x); \\
\text{print}(x)
\end{align*}
```

Before we begin, could you summarize the result of evaluating the function calls? What are the side effects (changes to memory and printed expressions)? At what steps do actual memory operations occur (dereferences)?

Note that the parser we have would convert the program above into the following form:

```latex
((\text{var } x: \text{Int}) \Rightarrow
((\text{pann } y: t) \Rightarrow y = y+1; y)(\text{print}(7); x); \\
\text{print}(x))(5)
```

Given the evaluation rule DoApplyFunVar, we get:

\[
\langle M = \{a_x \rightarrow 5\}, ((y: t) \Rightarrow y = y+1; y)(\text{print}(7); *a_x); \text{print}(*a_x) \rangle
\]

**Solution:**

For the \textbf{val} case, we do not really have mutation, so let’s consider the following configuration instead:

\[
\langle M = \{a_x \rightarrow 5\}, ((y: t) \Rightarrow y+1; y)(\text{print}(7); *a_x); \text{print}(*a_x) \rangle
\]

Evaluate argument to a value.

Print 7 to screen.

Perform dereference, retrieve value \(M(a_x) = 5\) from memory:

\[
\langle M = \{a_x \rightarrow 5\}, ((y: t) \Rightarrow y+1; y)(5); \text{print}(*a_x) \rangle
\]

Bind parameter to argument value:

\[
\langle M = \{a_x \rightarrow 5\}, (5+1; 5); \text{print}(*a_x) \rangle
\]

... Return value 5.

Perform dereference, retrieve value \(M(a_x) = 5\) from memory:

\[
\langle M = \{a_x \rightarrow 5\}, \text{print}(5) \rangle
\]
Print 5 to screen.
⟨M = \{a_x \rightarrow 5\}, ()⟩

Note that there was no memory mutation associated with this function call. Argument expression was evaluated to a value before passed into the function (before parameter bound to argument value). Result: \(M = \{a_x \rightarrow 5\}\), printed: 7, 5 to screen.

For the \texttt{var} case:
⟨M = \{a_x \rightarrow 5\}, ((y: t) \Rightarrow y = y+1; y)(\text{print}(7); *a_x); \text{print}(\text{*a}_x))⟩
Evaluate argument to a value.
Print 7 to screen.
⟨M = \{a_x \rightarrow 5\}, ((y: t) \Rightarrow y+1; y)(\text{print}(7); *a_x); \text{print}(\text{*a}_x))⟩
Perform dereference, retrieve value \(M(a_x) = 5\) from memory:
⟨M = \{a_x \rightarrow 5\}, ((y: t) \Rightarrow y+1; y)(5); \text{print}(\text{*a}_x))⟩
\text{DoApplFunVar} forces us to allocate a new memory address \(a_y\) that points to the value of the argument (5) and replace occurrences of parameter with \(*a_y\):
⟨M' = \{a_x \rightarrow 5, a_y \rightarrow 5\}, (*a_y = *a_y + 1; *a_y); \text{print}(\text{*a}_x))⟩
Perform dereference, retrieve value \(M'(a_y) = 5\) from memory:
⟨M' = \{a_x \rightarrow 5, a_y \rightarrow 5\}, (*a_y = 5 + 1; *a_y); \text{print}(\text{*a}_x))⟩
⟨M' = \{a_x \rightarrow 5, a_y \rightarrow 5\}, (*a_y = 6; *a_y); \text{print}(\text{*a}_x))⟩
Update memory to store \(M'(a_y) = 6\)
⟨M'' = \{a_x \rightarrow 5, a_y \rightarrow 6\}, (*a_y); \text{print}(\text{*a}_x))⟩
Perform dereference, retrieve and return value \(M''(a_y) = 6\) from memory:
⟨M'' = \{a_x \rightarrow 5, a_y \rightarrow 6\}. \text{print}(\text{*a}_x))⟩
Perform dereference, retrieve value \(M(a_x) = 5\) from memory:
⟨M'' = \{a_x \rightarrow 5, a_y \rightarrow 6\}. \text{print}(5)⟩
Print 5 to screen.
⟨M'' = \{a_x \rightarrow 5, a_y \rightarrow 6\}, ()⟩

Note that \text{DoApplyFunVar} requires the introduction of a new memory address to store the value of the argument to the function. Argument expression was evaluated to a value before passed into the function (before parameter bound to
argument value). Result: $M'' = \{a_x \to 5, a_y \to 6\}$, printed: 7, 5 to screen.

For the ref case:

$\langle M = \{a_x \to 5\}, ((y: t) \Rightarrow y = y+1; y)(\text{print}(7); *a_x); \text{print}(\ast a_x) \rangle$

Print 7 to screen.

$\langle M = \{a_x \to 5\}, ((y: t) \Rightarrow y = y+1; y)(*a_x); \text{print}(\ast a_x) \rangle$

Do not allocate new memory address, reuse argument’s address, perform substitution:

$\langle M = \{a_x \to 5\}, \ast a_x = \ast a_x+1; \ast a_x(5); \text{print}(\ast a_x) \rangle$

Perform dereference, retrieve value $M(a_x) = 5$ from memory:

$\langle M' = \{a_x \to 5\}, (*a_x = 5 + 1; *a_x); \text{print}(\ast a_x) \rangle$

$\langle M' = \{a_x \to 5\}, (*a_x = 6; *a_x); \text{print}(\ast a_x) \rangle$

Update memory to store $M''(a_x) = 6$:

$\langle M'' = \{a_x \to 6\}, (*a_x); \text{print}(\ast a_x) \rangle$

Perform dereference, retrieve and return value $M''(a_x) = 6$ from memory:

$\langle M'' = \{a_x \to 6\}, \text{print}(\ast a_x) \rangle$

Perform dereference, retrieve from memory and print value $M''(a_x) = 6$:

$\langle M'' = \{a_x \to 6\}, \text{print}(6) \rangle$

$\langle M'' = \{a_x \to 6\}, () \rangle$

Note that we do not allocate new memory addresses for function parameters, but reuse ones passed in as arguments. This is why argument expressions must resolve to Deref of an address. Technically, this program does not type check, but you could imagine dropping print(7) from the argument expression. The important point is that memory changes are global (updates to memory persists even after function call). Result: $M'' = \{a_x \to 6\}$.

What do you think the result of evaluating the following program is:

$\langle M = \{a_x \to 5\}, ((\text{name } y: t) \Rightarrow y = y+1; y)(\text{print}(7); *a_x); \text{print}(\ast a_x) \rangle$?
Solution: The program as is actually does not type check. This is the assignment expression in the body of the function requires the parameter\((y)\) to already be in the type environment and decorated as writable. Typing rule for function with \(pann = name\) (last rule in Figure 5, page 11 of handout) adds the parameter to the typing environment as read-only.

So, let’s tweak the example as in case of \(\text{val}\):
\[
\langle M = \{a \rightarrow 5\}, ((\text{name } y: t) \mapsto y+1; y)(\text{print}(7); \ast a_x); \text{print}(\ast a_x) \rangle
\]
Substitute argument expression for uses of parameter in the function body:
\[
\langle M = \{a \rightarrow 5\}, (\text{print}(7); \ast a_x+1; \text{print}(7); \ast a_x); \text{print}(\ast a_x) \rangle
\]
Print 7 to the screen.
Perform dereference, retrieve value \(M(a_x) = 5\) from memory.

... Print 7 again to the screen.
Perform dereference, retrieve and return value \(M(a_x) = 5\) from memory:
\[
\langle M = \{a \rightarrow 5\}, \text{print}(\ast a_x) \rangle
\]
Perform dereference, retrieve value \(M(a_x) = 5\) from memory:
\[
\langle M = \{a \rightarrow 5\}, \text{print}(5) \rangle
\]
Print 5 to the screen.
\[
\langle M = \{a \rightarrow 5\}, () \rangle
\]

Note we did not update memory. We re-evaluated the argument expression for every use of the parameter, so side-effects were doubled.

What would happen with a call-by-name evaluation strategy if we never use the parameter in the function body expression? Is this efficient?

Summary:
<table>
<thead>
<tr>
<th>Call-by-value</th>
<th>Call-by-reference</th>
<th>Call-by-name</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>val</strong></td>
<td><strong>var</strong></td>
<td><strong>ref</strong></td>
</tr>
<tr>
<td>Eval arg to value (side effects)</td>
<td>Eval arg to value (side effects)</td>
<td>Eval arg to Deref expression (side effects)</td>
</tr>
<tr>
<td>No memory mutation</td>
<td>New addr for fun param</td>
<td>“Reuse” arg addr for param</td>
</tr>
<tr>
<td>New addr not accessible after call</td>
<td>Memory changes are persistent</td>
<td></td>
</tr>
</tbody>
</table>