Lecture13-14
Procedural, Functional Abstractions and Parameter Passing

March 18, 2013

Control Abstractions: Parameter Passing

- Parameters: Where and Why
- Procedure and Function Invocation: In-depth
- By Value Parameter Passing: FUI
- By Name P. P. (command and expression): FUI
- By Need P. P.: Implementative variant
- By Function, Procedure P. P. (Closure Transmission): Implementative variant
- By reference P.P.: FUI
- By Constant P.P.: FUI
- By Result P.P.: FUI
- By Value-Result P.P.: FUI

Parameters: Where and Why

- **Where.** Anywhere an abstraction (i.e. generalization) is introduced, the problem of its use in possibly, many different contexts, must be considered;
- The use in a specific context requires an *instantiation* mechanism which intimately connects the abstraction to the context of use;
- **Why.**The connection is realized through the use of formal parameters (with which abstraction is made) and the actual parameters which express the context in which abstraction must be used;
- Then, the instantiation of the abstraction consists in the creation of the bindings which connect each formal parameter to the corresponding actual parameter.;
- The mechanism that creates this connection is called Parameter Passing (or P. Transmission).

Procedure and Function Invocation: In more depth

- Parameter Passing consists of 3 distinct steps:
  - **Transmission:** It evaluates the arguments (i.e. actual parameters) according to the kind of transmission that has been specified (in the corresponding formal parameter) and results into a list, the transmission list, of denotable or storable values, one for each argument;
  - **Binding:** It makes a binding between each formal parameter and the corresponding value of the transmission list. The effective form of the binding depends on the kind of transmission of the formal parameter;
  - **Return:** It binds back the values, computed by body execution, to the arguments that have been passed by one of various form of by result parameter passing. It results in various modifications of the store.

- Hence, the effective structure of the invoked code consists of:
  - Prologue: it includes code for the steps Binding;
  - Body: it corresponds to the body of the procedure or function
  - Epilogue: It includes code for the step Return

- Then, where is put the code for the step Transmission?

Procedure and Function Invocation: In more depth
The Structure of General Invocation

- **Q:** Then, where is put the code for the step Transmission?
- **A:** In the code for invocation, of course.

- Step **Transmission** is formally defined by **function** $\mathcal{T}$ in definitional tables.

Auxiliary Syntactic Domains

$$\begin{aligned} \mathtt{A} ::= {} & \mathtt{byValue\ E} \mid \mathtt{byName\ E} \mid \mathtt{byReference\ E} \\ & \mid \mathtt{byConst\ E} \mid \mathtt{byResult\ E} \mid \mathtt{byValueResult\ E} \end{aligned}$$

Semantic Functions

$$\mathcal{M}[\![\mathtt{Call\ I\ (A_1 \ldots A_n)}]\!]\rho : \mathtt{Store} \to \mathtt{Store}$$

$$\mathcal{T} : \mathtt{A}^n \to \mathtt{Env} \to \mathtt{Store} \to ((\mathtt{Val} \cup \mathtt{Den})^n \times \mathtt{Store})_\perp$$

$$\mathcal{M}[\![\mathtt{Call\ I\ (A_1 \ldots A_n)}]\!]\rho(\mathtt{s}) =$$
$$\mathtt{Let}\{((v_1 \ldots v_n), \mathtt{s_n}) = \mathcal{T}[\![(\mathtt{A_1 \ldots A_n})]\!]\rho(\mathtt{s}),\ \mathtt{F(f)} = \rho(\mathtt{I})\}$$
$$\mathtt{f}(v_1 \ldots v_n)(\mathtt{s_n})$$

Procedure and Function Invocation: In more depth
The Structure of General Declaration

- Step **Binding** is formally defined by **function** $\mathcal{B}$
- Step **Return** is formally defined by **function** $\mathcal{R}$
- $\mathcal{D}_E$ is for stressing that the declaration of a procedure is an invariant of the store

Auxiliary Syntactic Domains

$$\begin{aligned}&\mathtt{P} ::= \mathtt{byValue\ I} \mid \mathtt{byName\ I} \mid \mathtt{byReference\ I}\\ &\qquad \mid \mathtt{byConst\ I} \mid \mathtt{byResult\ I} \mid \mathtt{byValueResult\ I}\end{aligned}$$

Semantic Functions

$$\mathcal{D}[\![\mathtt{D}]\!] : \mathtt{Env} \to \mathtt{Store} \to (\mathtt{Env} \times \mathtt{Store})_\perp$$

$$\mathcal{D}_E[\![\mathtt{Proc\ I\ (P_1 ... P_n)\ C}]\!] : \mathtt{Env} \to \mathtt{Env}$$

$$\mathcal{D}[\![\mathtt{D}]\!]\rho = \lambda \mathtt{s}.(\mathcal{D}_E[\![\mathtt{D}]\!]\rho, \mathtt{s})$$

$$\mathcal{B} : \mathtt{P^n} \to (\mathtt{Env} \times (\mathtt{Val} \cup \mathtt{Den})^\mathtt{n} \times \mathtt{Store}) \to (\mathtt{Env} \times () \times \mathtt{Store})_\perp$$

$$\mathcal{R} : \mathtt{P^n} \to \mathtt{Env} \to ((\mathtt{Val} \cup \mathtt{Den})^\mathtt{n} \times \mathtt{Store}) \to \mathtt{Store}_\perp$$

$$\begin{aligned}&\mathcal{D}_E[\![\mathtt{Proc\ I\ (P_1 ... P_n)\ C}]\!]\rho =\\ &\quad \mathtt{Let}\{\mathtt{f} = \lambda(\mathtt{v_1 ... v_n}).\lambda \mathtt{s}.\mathtt{s_r}\\ &\qquad \mathtt{where}\{(\rho_\mathtt{n}, \_, \mathtt{s_n}) = \mathcal{B}[\![(\mathtt{P_1 ... P_n})]\!](\rho, (\mathtt{v_1 ... v_n}), \mathtt{s})\}\\ &\qquad\qquad \{\mathtt{s_c} = \mathcal{M}[\![\mathtt{C}]\!]\rho_\mathtt{n}(\mathtt{s_n})\}\\ &\qquad\qquad \{\mathtt{s_r} = \mathcal{R}[\![(\mathtt{P_1 ... P_n})]\!]\rho_\mathtt{n}((\mathtt{v_1 ... v_n}), \mathtt{s_c})\}\\ &\quad \mathtt{bind(I, F(f), \rho)}\end{aligned}$$

By Value Parameter Passing: FUI

By Value Parameter Passing : $\mathcal{T}, \mathcal{B}, \mathcal{R}$

Auxiliaries Semantic Functions

$$\mathcal{T}[\![(\mathtt{A}_1 \ldots \mathtt{A}_n)]\!]\rho(\mathtt{s}) = \mathcal{T}_1[\![\mathtt{A}_1]\!]\rho \circ \ldots \circ \mathcal{T}_1[\![\mathtt{A}_n]\!]\rho\ ((), \mathtt{s})$$

$$\mathcal{T}_1[\![\mathtt{byValue\ AMem(E)}]\!]\rho((v_1 \ldots v_m), \mathtt{s}_m) =$$
$$\mathtt{Let}\{(v_{m+1}, \mathtt{s}_{m+1}) = \perp_S \mathcal{E}[\![\mathtt{E}]\!]\rho(\mathtt{s}_m)\}((v_1 \ldots v_m \mathtt{VM}(v_{m+1})), \mathtt{s}_{m+1})$$

$$\mathcal{B}[\![(\mathtt{P}_1 \ldots \mathtt{P}_n)]\!]\rho((v_1 \ldots v_n), \mathtt{s}) = (\mathcal{B}_1[\![\mathtt{P}_1]\!] \circ \ldots \circ \mathcal{B}_1[\![\mathtt{P}_n]\!] \circ \downarrow^3_{1-3})(\rho, (v_1 \ldots v_n), \mathtt{s})$$

$$\mathcal{B}_1[\![\mathtt{byValue\ I}_k]\!](\rho_{k-1}, (v_k \ldots v_n), \mathtt{s}_{k-1}) =$$
$$\mathtt{Let}\{(\mathtt{l}_k, \mathtt{s}'_k) = \mathtt{allocate}(\mathtt{s}_{k-1})\}$$
$$\{\mathtt{s}_k = \mathtt{upd}(\mathtt{l}_k, \mathtt{v}_k, \mathtt{s}'_k), \rho_k = \mathtt{bind}(\mathtt{I}_k, \mathtt{l}_k, \rho_{k-1})\}$$
$$(\rho_k, (v_{k+1} \ldots v_n), \mathtt{s}_k)$$

$$\mathcal{R}[\![(\mathtt{P}_1 \ldots \mathtt{P}_n)]\!]\rho((v_1 \ldots v_n), \mathtt{s}) = (\mathcal{R}_1[\![\mathtt{P}_1]\!]\rho \circ \ldots \circ \mathcal{R}_1[\![\mathtt{P}_n]\!]\rho \circ \downarrow^2_2)((v_1 \ldots v_n), \mathtt{s})$$

$$\mathcal{R}_1[\![\mathtt{byValue\ I}_k]\!]\rho((v_k \ldots v_n), \mathtt{s}_{k-1}) =$$
$$\mathtt{Let}\{\mathtt{s}_k = \mathtt{s}_{k-1}\}((v_{k+1} \ldots v_n), \mathtt{s}_k)$$

- The actual parameter must be an expression whose evaluation must result a storable value: This is stressed by AMem(E)
- Binding creates a mutable value $\mathtt{l}_k$
- Return does nothing

By Value Parameter Passing: FUI /2

- Use.
  - It is the default parameter passing of almost all languages (Algol 60, Simula, Pascal, C/C++, ML, Ocaml, Ada, C#, Java)
  - It makes a One-way connection: The callee has a copy of the storable value in the caller context
  - It is used in some programming techniques, for passing values to the caller and for using the formal parameter as a mutable for temporary values or for an accumulator
  - No Side Effects in the(store of) caller context
- Implementation
  - Similar to that of the variable declaration with initialization

By Name Parameter Passing: FUI /1

By Value Parameter Passing : $\mathcal{T}, \mathcal{B}, \mathcal{R}$

Auxiliaries Semantic Functions

$$\mathcal{T}[\![(\mathtt{A}_1 \ldots \mathtt{A}_n)]\!]\rho(\mathtt{s}) = \mathcal{T}_1[\![\mathtt{A}_1]\!]\rho \circ \ldots \circ \mathcal{T}_1[\![\mathtt{A}_n]\!]\rho\ ((), \mathtt{s})$$
$$\mathcal{T}_1[\![\mathtt{byName\ ACodeC(C)}]\!]\rho((v_1 \ldots v_m), \mathtt{s}_m) = ((v_1 \ldots v_m \mathtt{Z}(\mathcal{M}[\![\mathtt{C}]\!]\rho)), \mathtt{s}_{m+1})$$
$$\mathcal{T}_1[\![\mathtt{byName\ ACodeE(E)}]\!]\rho((v_1 \ldots v_m), \mathtt{s}_m) = ((v_1 \ldots v_m \mathtt{Z}(\mathcal{E}[\![\mathtt{E}]\!]\rho)), \mathtt{s}_{m+1})$$

$$\mathcal{B}[\![(\mathtt{P}_1 \ldots \mathtt{P}_n)]\!]\rho((v_1 \ldots v_n), \mathtt{s}) = (\mathcal{B}_1[\![\mathtt{P}_1]\!] \circ \ldots \circ \mathcal{B}_1[\![\mathtt{P}_n]\!] \circ \downarrow^3_{1-3})(\rho, (v_1 \ldots v_n), \mathtt{s})$$
$$\mathcal{B}_1[\![\mathtt{byName\ I}_k]\!](\rho_{k-1}, (v_k \ldots v_n), \mathtt{s}_{k-1}) =$$
$$\mathtt{Let}\{\rho_k = \mathtt{bind}(\mathtt{I}_k, v_k, \rho_{k-1}), \mathtt{s}_k = \mathtt{s}_{k-1}\}$$
$$(\rho_k, (v_{k+1} \ldots v_n), \mathtt{s}_k)$$

$$\mathcal{R}[\![(\mathtt{P}_1 \ldots \mathtt{P}_n)]\!]\rho((v_1 \ldots v_n), \mathtt{s}) = (\mathcal{R}_1[\![\mathtt{P}_1]\!]\rho \circ \ldots \circ \mathcal{R}_1[\![\mathtt{P}_n]\!]\rho \circ \downarrow^2_2)((v_1 \ldots v_n), \mathtt{s})$$
$$\mathcal{R}_1[\![\mathtt{byName\ I}_k]\!]\rho((v_k \ldots v_n), \mathtt{s}_{k-1}) =$$
$$\mathtt{Let}\{\mathtt{s}_k = \mathtt{s}_{k-1}\}((v_{k+1} \ldots v_n), \mathtt{s}_k)$$

Auxiliaries Semantic Functions

$\mathtt{Z} : \mathtt{Code} \rightarrow \mathtt{Den}$ (*injection*)

- The actual parameter must be a Code: This is stressed by `ACodeC(C)` and `ACodeE(E)`
- Binding creates a binding between name of the formal parameter and the Code
- Return does nothing

By Name Parameter Passing: FUI /2

- The code, passed to the callee, is closed with the bindings (i.e. environment) of the caller;
- The code may be an expression (possibly, an anonymous function, *lambda astrazione*) in the scope of the environment of the caller;
- The code may be a denotable expression that is used from caller/callee for sharing a mutable value
- Hence, $\mathcal{E}[\![]\!]$ must be extended on the expressions $\mathtt{Z}(v)$ that are bound to formals

Semantic Functions

$$\mathcal{E}[\![\mathtt{Val(I)}]\!]_\rho(\mathtt{s}) = \begin{cases} \ldots \\ v(\mathtt{s}) & \text{if } \rho(\mathtt{I}) \equiv \mathtt{Z}(v), \\ & \quad \text{for } v \in \mathtt{Store} \to (\mathtt{Val} \times \mathtt{Store})_\perp \end{cases}$$

$$\mathcal{E}[\![\mathtt{Den(I)}]\!]_\rho(\mathtt{s}) = \begin{cases} \ldots \\ (\mathtt{l}, \mathtt{s}) & \text{if } (\rho(\mathtt{I}) \equiv \mathtt{Z}(\mathtt{l})), \mathtt{l} \in \mathtt{Loc} \end{cases}$$

Example

By using by name passing, in Algol 60, a code for the computation of expressions with summation like the one below:

$$z = y + 5 \sum_{n \le x \le m} (3x^2 - 5x + 2)$$

introduces a specific function for $\sum$ and invokes it, as an operator, in the expression.