Lecture12 Commands: Formalization, Use and Implementation

prof. Marco Bellia, Dip. Informatica, Università di Pisa

April 8, 2014

Commands: Formalization, Use and Implementation

- Commands: Where and Why
- Structured Programming Paradigm
- Simple, Compound and Structured commands
- Sequence Control Commands
- Control Abstractions

Commands: Where and Why

Commands

- **Are In** Procedural Languages and their Extensions (included OO: $C^{\#}$, Java, Scala,...)
- **Involve** State Control and Update:
  - Store Control: assignment, dynamic allocation/deallocation,
  - Sequence Control: Definition and Modification of command sequence
- **Lead To** Prescriptive Programming and Algorithm Definitions that specify how computation must proceed.

Structured Programming Paradigm

Commands are Constructs for State Control of kind

- **Explicit.**
  State Update: Assignment
  Control transfer: `goto`, `break`, `continue`, `return`
  - **Low Level Programming.** In wide use in Machine Language
  - **Structured Programming Paradigm** forbids the use of Unconstrained Transfers, i.e. UTs.
    - **High Level Languages** can avoid the use of UTs
    - **Combination** of Structured commands with UTs leads to complex and error prone, program behaviours.
- **Conditional.** `if`, `break`, `case`, `switch`, `cond`
- **Iterative.** Two kinds: Determined, Non Determined
- **Recursion.** Procedure, Functions (control abstractions)

Simple, Compound, Structured Commands

- **Simple** involve (only) expressions
- **Compound** involve (inner) commands
- **Structured** Compound commands with:
  only one enter point
  only one exit point

Example

An inline block with a compound command while, an expression, inner commands c1,...,ck

```
{...while E do {c1;...;ck};...}
```

Example

An inline block with a compound command while, an expression, inner commands c1,...,ck, a goto command.

```
{...while E do {c1;...;A:cj;...;ck};...; goto A;...}
```

– The command while is a non structured command.
– The fragment is not for Structured Programming

Commands: Assignment and Sequencing

Syntactic Domains

$$\begin{aligned}\texttt{C} ::= {} & \texttt{C C} \mid \texttt{E} = \texttt{E} \mid \texttt{IF E Then C Else C} \mid \texttt{Case E } (\texttt{E}^{+} \texttt{ :C})^{+} \\ & \mid \texttt{For I} = \texttt{E To E By E Do C} \\ & \mid \texttt{While E Do C} \mid \texttt{Proc I()C} \mid \texttt{Call I()} \mid \ldots\end{aligned}$$

Semantic Functions

$$\mathcal{M}[\![\texttt{C}]\!]_\rho : \texttt{Store} \to \texttt{Store}_\perp$$

$$\mathcal{M}[\![\texttt{C}_1\ \texttt{C}_2]\!]_\rho = \mathcal{M}[\![\texttt{C}_1]\!]_\rho \circ \mathcal{M}[\![\texttt{C}_2]\!]_\rho$$

$$\mathcal{M}[\![\texttt{E}_l = \texttt{E}_r]\!]_\rho =$$

$$\lambda \texttt{s}.\texttt{Let}\{(\texttt{v}_1, \texttt{s}_1) = \mathcal{E}[\![\texttt{E}_\texttt{r}]\!]_\rho(\texttt{s})\}$$

$$\{(\texttt{l}_2, \texttt{s}_2) = \mathcal{E}[\![\texttt{E}_\texttt{l}]\!]_\rho(\texttt{s}_1)\}\ \texttt{upd}(\texttt{l}_2, \texttt{VM}(\texttt{v}_1), (\texttt{s}_2))$$

Auxiliares

$$\texttt{VM} : \texttt{Val} \to \texttt{Mem}$$

Commands: Conditionals

Example

Complete.

Semantic Functions

$\mathcal{M}[\![\mathtt{C}]\!]_\rho : \mathtt{Store} \to \mathtt{Store}_\perp$

$\mathcal{M}[\![\mathtt{IF\ E\ Then\ C_1\ Else\ C_2}]\!]_\rho(\mathtt{s}) = \ldots$

$\mathcal{M}[\![\mathtt{Case\ E\ (E_1{:}C_1)\ldots(E_n{:}C_n)}]\!]_\rho(\mathtt{s}) = \ldots$

Commands: Determined Iterator

Syntactic Domains

$$\mathtt{C} ::= \ldots \mid \mathtt{For\ I = E\ To\ E\ By\ E\ Do\ C} \mid \ldots$$

Semantic Functions

$$\mathcal{M}[\![\mathtt{C}]\!]_\rho : \mathtt{Store} \to \mathtt{Store}_\perp$$

$$\mathcal{M}[\![\mathtt{For\ I = E_1\ To\ E_2\ By\ E_3\ Do\ C}]\!]_\rho(\mathtt{s}) =$$

$$\mathtt{Let}\{(\mathtt{inizio}, \mathtt{s_1}) = \mathcal{E}[\![\mathtt{E_1}]\!]_\rho(\mathtt{s})\}$$

$$\{(\mathtt{fine}, \mathtt{s_2}) = \mathcal{E}[\![\mathtt{E_2}]\!]_\rho(\mathtt{s_1})\}$$

$$\{(\mathtt{passo}, \mathtt{s_3}) = \mathcal{E}[\![\mathtt{E_3}]\!]_\rho(\mathtt{s_2})\}$$

$$\{\mathtt{n} = (\mathtt{fine} - \mathtt{inizio} + \mathtt{passo})/\mathtt{passo}, \mathtt{g} = \mathcal{M}[\![\mathtt{C}]\!]_\rho\}$$

$$\mathtt{g}^{\mathtt{n}}(\mathtt{s_3})$$

Commands: Non Determined Iterator

Syntactic Domains

$$\mathtt{C} ::= \ldots \mid \mathtt{While\ E\ Do\ C} \mid \ldots$$

Semantic Functions

$$\mathcal{M}[\![\mathtt{C}]\!]_\rho : \mathtt{Store} \to \mathtt{Store}_\perp$$

$$\mathcal{M}[\![\mathtt{While\ E\ Do\ C}]\!]_\rho =$$

$$\mathtt{Y\ g}.\lambda\ \mathtt{g}.\lambda \mathtt{s}.$$

$$\mathtt{Let}\{(\mathtt{v}, \mathtt{s}_1) = \mathcal{E}[\![\mathtt{E}]\!]_\rho(\mathtt{s})\}$$

$$\mathtt{if}(\mathtt{false}(\mathtt{v}), \mathtt{s}_1, (\mathcal{M}[\![\mathtt{C}]\!]_\rho \circ\ \mathtt{g})(\mathtt{s}_1))$$

Commands: Implementation

- Commands are decomposed into sequences of more elementary state control operations (3AC code, Bytecode or similar) for:
  - store and register update
  - state test
  - sequence control transfer (Program Counter reset)

Commands: Inline Block

Syntactic Domains

$\mathtt{C} ::= \ldots \mid \{\mathtt{D}\ \mathtt{C}\} \mid \ldots$

Semantic Functions

$\mathcal{M}[\![\{\mathtt{D}\ \mathtt{C}\}]\!]_\rho : \mathtt{Store} \to \mathtt{Store}_\perp$

$$\mathcal{M}[\![\{\mathtt{D}\ \mathtt{C}\}]\!]_\rho = \lambda \mathtt{s}.\mathtt{Let}\{(\rho_1, \mathtt{s}_1) = \mathcal{D}[\![\mathtt{D}]\!]_\rho(\mathtt{s})\} \mathcal{M}[\![\mathtt{C}]\!]_{\rho_1}(\mathtt{s}_1)$$

Example

This fragment now, is completely meaningful. Provide the semantics.

```
{var x=5; var y; y=x+2; {var y=x; x=y=z;} y=x;}
```

- Implementation
  - AR: One for each block;
  - Stack and AR templates, for recursive procedure block.

Control Abstractions

Syntactic Domains

$\mathtt{D} ::= \ldots \mid \mathtt{Function\ I(P_1\ Ide_1, \ldots, P_n\ Ide_n)E}$
$\mid \mathtt{Function\ I(P_1\ Ide_1, \ldots, P_n\ Ide_n)C} \mid \ldots$

$\mathtt{C} ::= \ldots \mid \mathtt{Proc\ I(P_1\ Ide_1, \ldots, P_n\ Ide_n)C} \mid \mathtt{Call\ I(A_1, \ldots, A_n)} \mid \ldots$

$\mathtt{E} ::= \ldots \mid \mathtt{A} \mid \mathtt{Call\ I(A_1, \ldots, A_n)} \mid \ldots - \mathtt{A_i\ are\ Actual\ Parameters}$

Semantic Functions

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

$\mathcal{D}[\![\mathtt{Proc\ I()C}]\!]_\rho(\mathtt{s}) =$

$\mathtt{Let}\{\mathtt{g} = \lambda().\lambda\mathtt{s}'.\mathcal{E}[\![\mathtt{E}]\!]_\rho(\mathtt{s}')\}\ (\mathtt{bind}(\mathtt{I}, \mathtt{F}(\mathtt{g}), \rho), \mathtt{s})$

$\mathcal{E}[\![\mathtt{E}]\!]_\rho : \mathtt{Store} \to (\mathtt{Val}_\perp \times \mathtt{Store}_\perp)$

$\mathcal{E}[\![\mathtt{Call\ I()}]\!]_\rho = \lambda().\lambda\mathtt{s}.\mathtt{Let}\{\mathtt{F}(\mathtt{g}) = \rho(\mathtt{I})\}\ \mathtt{g}()(\mathtt{s})$

Auxiliares

$\mathtt{F} : (\mathtt{State} \to (\mathtt{Val} \times \mathtt{State})) \to \mathtt{ProcFun}$

Control Abstractions: Use

- Use
  - **Localization** of the code of a functionality of the implemented algorithm;
  - **Modification and Certification** of the code that is embodied in a Programming Unit;
  - **Reuse** of the code that is embodied in Programming Unit;
  - **Recursion** (and inductive algorithms) is supported through Control Abstractions
  - Functional Values are fundamental in Higher Order Programming

- Suggetsed Reading:
  Gabrielli M., S. Martini, Programming Languages: Principles and Paradigms, Springer, 2006 - Chapter 6, pp 119-163.