Declarative Computation Model

Kernel language semantics

(Non-)Suspendable statements (VRH 2.4.3-2.4.4)

Carlos Varela
RPI
March 18, 2010

Adapted with permission from:
Seif Haridi
KTH
Peter Van Roy
UCL
Sequential declarative computation model

- The kernel language semantics
  - The environment: maps textual variable names (variable identifiers) into entities in the store
  - Abstract machine consists of an execution stack of semantic statements transforming the store
  - Interpretation (execution) of the kernel language elements (statements) by the use of an abstract machine
    - Non-suspendable statements
    - Suspendable statements
The following defines the syntax of a statement, \langle s \rangle denotes a statement

\[
\langle s \rangle ::= \text{skip} \\
| \quad \langle x \rangle = \langle y \rangle \\
| \quad \langle x \rangle = \langle v \rangle \\
| \quad \langle s_1 \rangle \langle s_2 \rangle \\
| \quad \text{local} \langle x \rangle \ \text{in} \ \langle s_1 \rangle \ \text{end} \\
| \quad \text{if} \langle x \rangle \ \text{then} \langle s_1 \rangle \ \text{else} \langle s_2 \rangle \ \text{end} \\
| \quad \{ \langle x \rangle \langle y_1 \rangle \ldots \langle y_n \rangle \} \\
| \quad \text{case} \langle x \rangle \ \text{of} \langle \text{pattern} \rangle \ \text{then} \langle s_1 \rangle \ \text{else} \langle s_2 \rangle \ \text{end} \\
\]

\[
\langle v \rangle ::= \text{proc} \ \{ \$ \langle y_1 \rangle \ldots \langle y_n \rangle \} \ \langle s_1 \rangle \ \text{end} \ | \ldots \\
\]

\[
\langle \text{pattern} \rangle ::= \ldots \\
\]

empty statement
variable-variable binding
variable-value binding
sequential composition
declaration
conditional
procedural application
pattern matching
value expression
A computation defines how the execution state is transformed step by step from the initial state to the final state.

A single assignment store $\sigma$ is a set of store variables, a variable may be unbound, bound to a partial value, or bound to a group of other variables.

An environment $E$ is mapping from variable identifiers to variables or values in $\sigma$, e.g. $\{X \rightarrow x_1, Y \rightarrow x_2\}$.

A semantic statement is a pair $(\langle s \rangle, E)$ where $\langle s \rangle$ is a statement.

$ST$ is a stack of semantic statements.
Computations (abstract machine)

• A computation defines how the execution state is transformed step by step from the initial state to the final state.

• The *execution state* is a pair
  
  \[(ST, \sigma)\]

  where \(ST\) is a *stack of semantic statements* and \(\sigma\) is a *single assignment store*.

• A *computation* is a sequence of execution states
  
  \[(ST_0, \sigma_0) \rightarrow (ST_1, \sigma_1) \rightarrow (ST_2, \sigma_2) \rightarrow ...\]
Semantics

- To execute a program (i.e., a statement) \( \langle s \rangle \) the initial execution state is
  \[
  ( [ \langle s \rangle, \varnothing ] , \varnothing )
  \]
- \( ST \) has a single semantic statement \( (\langle s \rangle, \varnothing) \)
- The environment \( E \) is empty, and the store \( \sigma \) is empty
- \( [ ... ] \) denotes the stack
- At each step the first element of \( ST \) is popped and execution proceeds according to the form of the element
- The final execution state (if any) is a state in which \( ST \) is empty
• The semantic statement is $(\text{skip}, E)$
• Continue to next execution step

\begin{align*}
\begin{array}{c}
\text{(skip, } E) \\
\text{ST}
\end{array} + \begin{array}{c}
\sigma
\end{array} & \rightarrow & \begin{array}{c}
\text{ST}
\end{array} + \begin{array}{c}
\sigma
\end{array}
\end{align*}
Sequential composition

- The semantic statement is $(\langle s_1 \rangle \langle s_2 \rangle , E)$
- Push $(\langle s_2 \rangle , E)$ and then push $(\langle s_1 \rangle , E)$ on $ST$
- Continue to next execution step

\[
\begin{array}{c|c|c}
(\langle s_1 \rangle \langle s_2 \rangle , E) & (\langle s_1 \rangle , E) & (\langle s_2 \rangle , E) \\
ST & ST & ST \\
\end{array}
\]
Variable declaration

• The semantic statement is
  \((\text{local } \langle x \rangle \text{ in } \langle s \rangle \text{ end, } E)\)

• Create a new store variable \(x\) in the Store

• Let \(E'\) be \(E + \{\langle x \rangle \rightarrow x\}\), i.e. \(E'\) is the same as \(E\) but the identifier \(\langle x \rangle\) is mapped to \(x\).

• Push \((\langle s \rangle, E')\) on \(ST\)

• Continue to next execution step
Variable declaration

• The semantic statement is
\((\text{local } X \text{ in } \langle s \rangle \text{ end}, E)\)
Variable-variable equality

- The semantic statement is
  \((\langle x \rangle = \langle y \rangle, E)\)
- Bind \(E(\langle x \rangle)\) and \(E(\langle y \rangle)\) in the store
Variable-value equality

• The semantic statement is
  \(( \langle x \rangle = \langle v \rangle, E )\)
• Where \(\langle v \rangle\) is a record, a number, or a procedure
• Construct the value in the store and refer to it by the variable \(y\).
• Bind \(E(\langle x \rangle)\) and \(y\) in the store
• We have seen how to construct records and numbers, but what is a procedure value?
Procedure values

• Constructing a procedure value in the store is not simple because a procedure may have external references

local P Q in
  Q = proc ${} \{ \text{Browse hello} \} \text{ end}
  P = proc ${} \{ Q \} \text{ end}
local Q in
  Q = proc ${} \{ \text{Browse hi} \} \text{ end}
  \{ P \}
end
end
local P Q in
  Q = proc {$} {Browse hello} end
  P = proc {$} {Q} end
local Q in
  Q = proc {$} {Browse hi} end
  {P}
end
end

proc {$} {Q} end

P \rightarrow x_1

Q \rightarrow x_2

proc {$} {Browse hello} end

Browse \rightarrow x_0

C. Varela; Adapted w/permission from S. Haridi and P. Van Roy
Procedure values (3)

- The semantic statement is
  \[
  \langle x \rangle = \text{proc} \{ \langle y_1 \rangle \ldots \langle y_n \rangle \} \langle s \rangle \text{ end, } E
  \]
- \langle y_1 \rangle \ldots \langle y_n \rangle \text{ are the (formal) parameters of the procedure}
- Other free identifiers in \langle s \rangle \text{ are called external references } \langle z_1 \rangle \ldots \langle z_k \rangle
- These are defined by the environment \( E \) where the procedure is declared (lexical scoping)
- The contextual environment of the procedure \( CE \) is \( E \mid \{ \langle z_1 \rangle \ldots \langle z_k \rangle \} \)
- When the procedure is called \( CE \) is used to construct the environment for execution of \langle s \rangle

\[
\begin{align*}
\text{proc} \{ \langle y_1 \rangle \ldots \langle y_n \rangle \} \\
\langle s \rangle \\
\text{end }, \\
CE
\end{align*}
\]
Procedure values (4)

- Procedure values are pairs:
  \[(\text{proc } \{$ \langle v_1 \rangle \ldots \langle v_n \rangle \} \langle s \rangle \text{ end } , CE)\]
- They are stored in the store just as any other value

C. Varela; Adapted w/permission from S. Haridi and P. Van Roy
Procedure introduction

• The semantic statement is

\[
(\langle x \rangle = \text{proc} \{\langle y_1 \rangle \ldots \langle y_n \rangle\} \langle s \rangle \text{ end}, E)
\]

• Create a contextual environment:

\[
CE = E \mid \{\langle z_1 \rangle \ldots \langle z_k \rangle\}
\]

where \(\langle z_1 \rangle \ldots \langle z_k \rangle\) are external references in \(\langle s \rangle\).

• Create a new procedure value of the form:

\[
(\text{proc} \{\langle y_1 \rangle \ldots \langle y_n \rangle\} \langle s \rangle \text{ end}, CE), \text{ refer to it by the variable } x_p
\]

• Bind the store variable \(E(\langle x \rangle)\) to \(x_p\)

• Continue to next execution step
Suspendable statements

- The remaining statements require $\langle x \rangle$ to be bound in order to execute
- The activation condition ($E(\langle x \rangle)$ is determined), is that $\langle x \rangle$ be bound to a number, a record or a procedure value

$\langle s \rangle ::= \ldots$

| if $\langle x \rangle$ then $\langle s_1 \rangle$ else $\langle s_2 \rangle$ end | conditional |
| \{ $\langle x \rangle \langle y_1 \rangle \ldots \langle y_n \rangle$ \} | procedural application |
| case $\langle x \rangle$ of |
| $\langle \text{pattern} \rangle$ then $\langle s_1 \rangle$ |
| else $\langle s_2 \rangle$ end | pattern matching |
Life cycle of a thread

- **Running**
  - \( A \& B \) / Execute
  - \( A \& \) not \( B \) / Suspend
  - not \( A \) / Terminate

- **Suspended**
  - B / Resume

- **Terminated**
  - ST not empty

- **Top(ST) activation condition is true**

C. Varela; Adapted w/permission from S. Haridi and P. Van Roy
Conditional

• The semantic statement is
  \( ( \text{if } \langle x \rangle \text{ then } \langle s_1 \rangle \text{ else } \langle s_2 \rangle \text{ end }, E ) \)

• If the activation condition \( (E(\langle x \rangle)) \) is determined) is true:
  – If \( E(\langle x \rangle) \) is not Boolean (true, false), raise an error
  – \( E(\langle x \rangle) \) is true, push \( (\langle s_1 \rangle, E) \) on the stack
  – \( E(\langle x \rangle) \) is false, push \( (\langle s_2 \rangle, E) \) on the stack

• If the activation condition \( (E(\langle x \rangle)) \) is determined) is false:
  – Suspend
Procedure application

- The semantic statement is
  \[\{ \langle x \rangle \langle y_1 \rangle \ldots \langle y_n \rangle \}, E\]

- If the activation condition \(E(\langle x \rangle)\) is determined) is true:
  - If \(E(\langle x \rangle)\) is not a procedure value, or it is a procedure with arity that is not equal to \(n\), raise an error
  - If \(E(\langle x \rangle)\) is \(\langle \text{proc } \{$ \langle z_1 \rangle \ldots \langle z_n \rangle\} \langle s \rangle \text{ end}, CE \rangle\), push
    \[\langle s \rangle, CE + \{ \langle z_1 \rangle \rightarrow E(\langle y_1 \rangle) \ldots \langle z_n \rangle \rightarrow E(\langle y_n \rangle) \}\]
    on the stack

- If the activation condition \(E(\langle x \rangle)\) is determined) is false:
  - Suspend
Case statement

• The semantic statement is
  ( case \( \langle x \rangle \) of \( \langle l \rangle \) (\( \langle f_1 \rangle : \langle x_1 \rangle \) … \( \langle f_n \rangle : \langle x_n \rangle \))
    then \( \langle s_1 \rangle \)
    else \( \langle s_2 \rangle \) end, \( E \))

• If the activation condition (\( E(\langle x \rangle) \) is determined) is true:
  – If \( E(\langle x \rangle) \) is a record, and the label of \( E(\langle x \rangle) \) is \( \langle l \rangle \) and its arity
    is \([\langle f_1 \rangle \) … \( \langle f_n \rangle]\):
      push (local \( \langle x_1 \rangle = \langle x \rangle \). \( \langle f_1 \rangle \) … \( \langle x_n \rangle = \langle x \rangle \). \( \langle f_n \rangle \) in \( \langle s_1 \rangle \) end, \( E \))
      on the stack
  – Otherwise, push (\( \langle s_2 \rangle \), \( E \)) on the stack

• If the activation condition (\( E(\langle x \rangle) \) is determined) is false:
  – Suspend
Execution examples

local Max C in

proc {Max X Y Z}
  if X >= Y then Z=X else Z=Y end
end

{Max 3 5 C}
end
Execution examples (2)

• Initial state ([(⟨s⟩₁, ∅)], ∅)

• After local Max C in ...
  ( [(⟨s⟩₂, {Max → m, C → c})], {m, c} )

• After Max binding
  ( [(⟨s⟩₄, {Max → m, C → c})],
    {m = (proc{§ X Y Z} ⟨s⟩₃ end , ∅) , c} )
Execution examples (3)

- After Max binding
  ( $(\langle s \rangle_4, \{\text{Max} \rightarrow m, \text{C} \rightarrow c\})$, 
  \{m = (\text{proc}\{X Y Z\} \langle s \rangle_3 \text{end} , \emptyset) , c\} )

- After procedure call
  ( $(\langle s \rangle_3, \{X \rightarrow t_1, Y \rightarrow t_2, Z \rightarrow c\}) ]$, 
  \{m = (\text{proc}\{X Y Z\} \langle s \rangle_3 \text{end} , \emptyset) , t_1=3, t_2=5, c\} )

C. Varela; Adapted w/permission from S. Haridi and P. Van Roy
Execution examples (4)

• After procedure call
  \[
  [([⟨s⟩₃, \{X → t₁, Y → t₂, Z → c\}]),
  \{m = (proc\{X Y Z\} ⟨s⟩₃ end, ∅), t₁=3, t₂=5, c\})] \]

• After \(T = (X≥Y)\)
  \[
  [([⟨s⟩₃, \{X → t₁, Y → t₂, Z → c, T → t\}]),
  \{m = (proc\{X Y Z\} ⟨s⟩₃ end, ∅), t₁=3, t₂=5, c, t=false\})] \]

• \([([Z=Y, \{X → t₁, Y → t₂, Z → c, T → t\}]),
  \{m = (proc\{X Y Z\} ⟨s⟩₃ end, ∅), t₁=3, t₂=5, c, t=false\})] \]
Execution examples (5)

\[ \left\{ \begin{array}{l}
\langle s \rangle_1 \\
\langle s \rangle_2 \\
\langle s \rangle_3 \\
\langle s \rangle_4 \\
\end{array} \right. \\
\text{local Max C in} \\
\quad \text{proc} \{ \text{Max X Y Z} \} \\
\quad \quad \text{if } X \geq Y \text{ then } Z=X \text{ else } Z=Y \text{ end} \\
\quad \text{end} \\
\quad \{ \text{Max 3 5 C} \} \\
\quad \text{end} \\
\]
46. Does dynamic binding require keeping an environment in a closure (procedure value)? Why or why not?

47. VRH Exercise 2.9.2 (page 107)

48. After translating the following function to the kernel language:

   ```
   fun {AddList L1 L2}
   case L1 of H1|T1 then
      case L2 of H2|T2 then
         H1+H2|{AddList T1 T2}
      end
   else nil end
   end
   ```

   Use the operational semantics to execute the call

   ```
   {AddList [1 2] [3 4]}
   ```