Declarative Computation Model

Kernel language semantics
(Non-)Suspendable statements (CTM 2.4.3-2.4.5)

Carlos Varela
RPI
March 12, 2015

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
Kernel language syntax

The following defines the syntax of a statement, \( \langle s \rangle \) denotes a statement

\[
\langle s \rangle ::= \begin{align*}
\text{skip} & \quad \text{empty statement} \\
\langle x \rangle = \langle y \rangle & \quad \text{variable-variable binding} \\
\langle x \rangle = \langle v \rangle & \quad \text{variable-value binding} \\
\langle s_1 \rangle \langle s_2 \rangle & \quad \text{sequential composition} \\
\text{local} \langle x \rangle \text{ in } \langle s_1 \rangle \text{ end} & \quad \text{declaration} \\
\text{if} \langle x \rangle \text{ then } \langle s_1 \rangle \text{ else } \langle s_2 \rangle \text{ end} & \quad \text{conditional} \\
\{ \langle x \rangle \langle y_1 \rangle \ldots \langle y_n \rangle \} & \quad \text{procedural application} \\
\text{case} \langle x \rangle \text{ of } \langle \text{pattern} \rangle \text{ then } \langle s_1 \rangle \text{ else } \langle s_2 \rangle \text{ end} & \quad \text{pattern matching}
\end{align*}
\]

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

\[
\langle \text{pattern} \rangle ::= \ldots
\]
Computations (abstract machine)

- 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

  \(( ST_0 , \sigma_0 ) \rightarrow ( ST_1 , \sigma_1 ) \rightarrow ( ST_2 , \sigma_2 ) \rightarrow \ldots \)
Semantics

• To execute a program (i.e., a statement) $\langle s \rangle$ the initial execution state is
  \[
  ([ (\langle s \rangle, \emptyset) ] , \emptyset )
  \]
• $ST$ has a single semantic statement $(\langle s \rangle, \emptyset)$
• 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
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

\[
\begin{array}{c|c}
\langle s_1 \rangle \langle s_2 \rangle , E & \sigma \\
ST & \rightarrow
\end{array}
\quad
\begin{array}{c|c}
\langle s_1 \rangle , E & \sigma \\
\langle s_2 \rangle , E & \rightarrow
\end{array}
\quad
\begin{array}{c|c}
\sigma
\end{array}
\]
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?
Lexical scoping

- Free and bound identifier occurrences
- An identifier occurrence is *bound* with respect to a statement \(s\) if it is in the scope of a declaration inside \(s\)
- A variable identifier is declared either by a ‘local’ statement, as a parameter of a procedure, or implicitly declared by a case statement
- An identifier occurrence is *free* otherwise
- In a running program every identifier is bound (i.e., declared)
Lexical scoping (2)

- \( \text{proc} \ \{ P \ X \} \)
  - \( \text{local} \ Y \ \text{in} \ Y = 1 \ \{ \text{Browse} \ Y \} \ \text{end} \)
  - \( X = Y \)
  - \( \text{end} \)

Free Occurrences

Bound Occurrences
Lexical scoping (3)

- `local Arg1 Arg2 in`
  
  `Arg1 = 111*111`
  `Arg2 = 999*999`
  `Res = Arg1*Arg2`

end

This is not a runnable program!

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

- `local Res in`
  
  `local Arg1 Arg2 in`
  
  `Arg1 = 111*111`
  
  `Arg2 = 999*999`
  
  `Res = Arg1*Arg2`

  `end`

  `{Browse Res}`

  `end`
Lexical scoping (5)

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

P → x_1
    ( , , )

proc {$} {Q} end

Q → x_2

x_2 ( , , )

proc {$} {Browse hi} end

Browse → x_0

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

- The semantic statement is
  \((\langle x \rangle = \text{proc} \{ \langle y_1 \rangle \ldots \langle y_n \rangle \} \langle s \rangle \text{ end}, E)\)
- \(\langle y_1 \rangle \ldots \langle y_n \rangle\) are the (formal) parameters of the procedure
- Other free identifiers in \(\langle s \rangle\) 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 \(\langle s \rangle\)
Procedure values (4)

- Procedure values are pairs:
  \[(\mathtt{proc}\ \{\langle y_1 \rangle \ldots \langle y_n \rangle \}\ \langle s \rangle \ \mathtt{end},\ CE)\]

- They are stored in the store just as any other value

\[(\mathtt{proc}\ \{\langle y_1 \rangle \ldots \langle y_n \rangle \}\ \langle s \rangle \ \mathtt{end},\ CE)\]
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:
  \(CE = E |_{\{z_1 \ldots z_k\}}\) 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:
  \((\text{proc } \{\$ \langle y_1 \rangle \ldots \langle y_n \rangle\} \langle s \rangle \text{ end}, CE)\), 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

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

- **Running**:
  - $A \& B$: Execute
  - $A \& \text{not } B$: Suspend
  - not $A$: Terminate

- **Suspended**: $B$: Resume

- **Terminated**: $ST$ not empty
  - Top($ST$) activation condition is true
  - $A$
  - $B$
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 \(\text{proc} \{\text{\$} \langle z_1 \rangle \ldots \langle z_n \rangle\} \langle s \rangle \text{ end, } CE\), 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
  \[( \text{case } \langle x \rangle \text{ of } \langle l \rangle (\langle f_1 \rangle : \langle x_1 \rangle \ldots \langle f_n \rangle : \langle x_n \rangle) \]
  \[\text{then } \langle s_1 \rangle \]
  \[\text{else } \langle s_2 \rangle \text{ end , } E)\]

• If the activation condition \(E(\langle x \rangle)\) is determined) is true:
  – If \(E(\langle x \rangle)\) is a record, and the label of \(E(\langle x \rangle)\) is \(\langle l \rangle\) and its arity is \([\langle f_1 \rangle \ldots \langle f_n \rangle]\):
    push \((\langle s_1 \rangle, E + \{\langle x_1 \rangle \rightarrow E(\langle x \rangle).\langle f_1 \rangle, \ldots, \langle x_n \rangle \rightarrow E(\langle x \rangle).\langle f_n \rangle\})\) 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 example

local Max C in
  fun {Max X Y}
    if X >= Y then X else Y end
  end
  C = {Max 3 5}
end
Example in kernel language

\[
\begin{aligned}
\langle S \rangle & \text{ local Max in} \\
\langle S \rangle_1 & \text{ local A in} \\
\langle S \rangle_2 & \text{ local B in} \\
\langle S \rangle_3 & \text{ local C in} \\
\langle S \rangle_4 & \text{ Max = proc } \{X Y Z\} \\
& \text{ local T in} \\
& \text{ T = (X >= Y)} \\
& \text{ if T then } Z=X \text{ else } Z=Y \text{ end} \\
& \text{ end} \\
A &= 3 \\
B &= 5 \\
\{\text{Max A B C}\} & \text{ end} \\
\text{ end} \\
\text{ end} \\
\end{aligned}
\]
Execution example (2)

• Initial state: ( [([s], {}), {} )

• After four local … in
  ( [([s]_1, {Max \rightarrow m, A \rightarrow a, B \rightarrow b, C \rightarrow c})], {m, a, b, c} )

• After bindings of Max, A, B:
  ( [([s]_2, {Max \rightarrow m, A \rightarrow a, B \rightarrow b, C \rightarrow c})],
  \{m = (\text{proc}\{\$ X Y Z\} \langle s \rangle_3 \text{ end}, \emptyset), a = 3, b = 5, c\} )

• After \{Max A B C\} procedure call:
  ( [([s]_3, \{X \rightarrow a, Y \rightarrow b, Z \rightarrow c\}) ],
  \{m = (\text{proc}\{\$ X Y Z\} \langle s \rangle_3 \text{ end}, \emptyset), a = 3, b = 5, c\} )
Execution example (3)

• After $T = (X \geq Y)$:
  
  ( [(⟨$⟩_4, \{X \rightarrow a, Y \rightarrow b, Z \rightarrow c, T \rightarrow t\}) ],
   \{m = (\text{proc}\{X Y Z\} \langle s \rangle_3 \text{end} , \emptyset) , a = 3, b = 5, c, t = \text{false}\} )

• After the execution of the conditional statement:
  
  ( [(Z= Y , \{X \rightarrow a, Y \rightarrow b, Z \rightarrow c, T \rightarrow t\}) ],
   \{m = (\text{proc}\{X Y Z\} \langle s \rangle_3 \text{end} , \emptyset) , a = 3, b = 5, c, t=\text{false}\} )

• After the variable-variable binding:
  
  ( [ ],
   \{m = (\text{proc}\{X Y Z\} \langle s \rangle_3 \text{end} , \emptyset) , a = 3, b = 5, c = 5, t=\text{false}\} )

We end execution since the statement stack is empty. The store variable $c$ is effectively bound to $5$ (i.e., $\{\text{Max 3 5}\}$) in the store.
Procedures with external references

\[ \langle s \rangle_1 \left\{ \langle s \rangle_2 \left\{ \langle s \rangle_3 \right. \right. \]

local LB Y C in
Y = 5
proc \{LB X Z\}
if X >= Y then Z=X else Z=Y end
end
\{LB 3 C\}
end
Procedures with external references

\[
\langle s \rangle_1 \left\{ \begin{array}{l}
\text{local LB Y C in} \\
Y = 5 \\
\text{proc } \{LB X Z\} \\
\langle s \rangle_3 \text{ if } X \geq Y \text{ then } Z = X \text{ else } Z = Y \text{ end} \\
\{LB 3 \ C\} \\
\text{end} \\
\end{array} \right.
\]

- The procedure value of LB is 
- \((\text{proc}\{\$ X Z\} \langle s \rangle_3 \text{ end}, \{Y \rightarrow y\})\)
- The store is \(\{y = 5, \ldots\}\)
Procedures with external references

\[
\begin{aligned}
&\text{local } LB \ Y \ C \ \text{in} \\
&\quad Y = 5 \\
&\quad \text{proc } \{LB X Z\} \\
&\quad \quad \text{if } X \geq Y \text{ then } Z = X \text{ else } Z = Y \text{ end} \\
&\quad \text{end} \\
&\{LB \ 3 \ C\} \\
\end{aligned}
\]

- The procedure value of LB is 
- \((\text{proc}\{\$ \ X \ Z\} \ \langle s \rangle_3 \ \text{end}, \ \{Y \rightarrow y\})\)
- The store is \(\{y = 5, \ldots\}\)
- STACK: [( \{LB T C\} , \{Y \rightarrow y\} , LB \rightarrow lb, C \rightarrow c, T \rightarrow t\}) ]
- STORE: \{y = 5, lb = (\text{proc}\{\$ \ X \ Z\} \ \langle s \rangle_3 \ \text{end}, \ \{Y \rightarrow y\}) , t = 3, c\}
Procedures with external references

\[
\begin{align*}
\langle s \rangle_1 & \quad \langle s \rangle_2 \\
& \quad \text{local LB Y C in} \\
& \quad Y = 5 \\
& \quad \text{proc } \{\text{LB X Z}\} \\
& \quad \langle s \rangle_3 \\
& \quad \text{if } X \geq Y \text{ then } Z = X \text{ else } Z = Y \text{ end} \\
& \quad \text{end} \\
& \quad \{\text{LB 3 C}\} \\
& \quad \text{end}
\end{align*}
\]

- STACK: [\((\{\text{LB T C}\}, \{Y \to y, \text{LB} \to lb, C \to c, T \to t\})\) ]
- STORE: \(\{y = 5, lb = (\text{proc } \{\text{X Z}\} \langle s \rangle_3 \text{ end }, \{Y \to y\}) , t = 3, c\}\)
- STACK: [\((\langle s \rangle_3, \{Y \to y, X \to t, Z \to c\})\) ]
- STORE: \(\{y = 5, lb = (\text{proc } \{\text{X Z}\} \langle s \rangle_3 \text{ end }, \{Y \to y\}) , t = 3, c\}\)
Procedures with external references

\[
\begin{align*}
\langle s \rangle_1 & \left\{ \begin{array}{l}
\text{local LB Y C in} \\
\quad Y = 5 \\
\quad \text{proc} \{ LB X Z \} \\
\quad \text{if } X \geq Y \text{ then } Z = X \text{ else } Z = Y \text{ end} \\
\quad \{ LB 3 C \} \\
\end{array} \right. \\
\langle s \rangle_2 & \\
\langle s \rangle_3 & \\
\end{align*}
\]

- STACK: \([((s)_3, \{ Y \to y , X \to t , Z \to c \}) ]\]
- STORE: \{ y = 5, lb = (\text{proc}\{ X Z \} \langle s \rangle_3 \text{ end} , \{ Y \to y \}) , t = 3, c\}
- STACK: \([(Z=Y , \{ Y \to y , X \to t , Z \to c \}) ]\]
- STORE: \{ y = 5, lb = (\text{proc}\{ X Z \} \langle s \rangle_3 \text{ end} , \{ Y \to y \}) , t = 3, c\}
- STACK: [ ]
- STORE: \{ y = 5, lb = (\text{proc}\{ X Z \} \langle s \rangle_3 \text{ end} , \{ Y \to y \}) , t = 3, c = 5\}
Try the following!

```prolog
local LB Y C in
  Y = 5
  proc {LB X Z}
    if X >= Y then Z=X else Z=Y end
  end
local Y in
  Y = 10
  {LB 3 C}
end
end
```

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

46. Does dynamic binding require keeping an environment in a closure (procedure value)? Why or why not?

47. CTM Exercise 2.9.2 (page 107)

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

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

Use the operational semantics to execute the call

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

49. CTM Exercise 2.9.3 (page 107).

50. CTM Exercise 2.9.7 (page 108) – translate example to kernel language and execute using operational semantics.

51. Write an example of a program that suspends. Now, write an example of a program that never terminates. Use the operational semantics to prove suspension or non-termination.