Declarative Computation Model

Kernel language semantics revisited (VRH 2.4.5)
From kernel to practical language (VRH 2.6)
Exceptions (VRH 2.7)

Carlos Varela
RPI
March 19, 2012

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

- The **kernel language semantics revisited**.
  - Suspendable statements:
    - if,
    - case,
    - procedure application.
  - Procedure values
  - Procedure introduction
  - Procedure application.
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
Case statement

- The semantic statement is
  \[
  \text{( case } \langle x \rangle \text{ of } \langle l \rangle (\langle f_1 \rangle : \langle x_1 \rangle \ldots \langle f_n \rangle : \langle x_n \rangle )
  \]
  \[
  \begin{array}{c}
  \text{then } \langle s_1 \rangle \\
  \text{else } \langle s_2 \rangle \text{ end } , E
  \end{array}
  \]
- If the activation condition \((E(\langle x \rangle))\) is determined) is true:
  - If \(E(\langle x \rangle)\) is a record, the label of \(E(\langle x \rangle)\) is \(\langle l \rangle\) and
    its arity is \([\langle f_1 \rangle \ldots \langle f_n \rangle]\), and \(\langle x_1 \rangle \ldots \langle x_n \rangle\) are independent:
      - push \((\text{local } \langle x_1 \rangle = \langle x \rangle. \langle f_1 \rangle \ldots \langle x_n \rangle = \langle x \rangle. \langle f_n \rangle \text{ in } \langle s_1 \rangle \text{ end, } E)\)
        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
Procedure values

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

```lisp
local P Q in
    Q = proc ${} {Browse hello} end
P = proc ${} {Q} end
local Q in
    Q = proc ${} {Browse hi} end
    {P}
end
end
```
Procedure values (2)

```
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
```

```
proc {$} {Browse hello} end
```

```
Browse → x_0
```

```
x_2 → (\ , \ )
```

```
x_1 → (\ , \ )
```

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 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:
  \[CE = E | \{\langle z_1 \rangle \ldots \langle z_k \rangle\}\]
  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:
  \[(\text{proc}\{\langle y_1 \rangle \ldots \langle y_n \rangle\}\langle s \rangle\text{ end}, CE)\]
  refer to it by the variable \(x_P\)
- Bind the store variable \(E(\langle x \rangle)\) to \(x_P\)
- Continue to next execution step
Procedure application

- The semantic statement is
  \( (\{x\}, y_1 \ldots y_n, E) \)
- 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 \( \text{proc} \{ z_1 \ldots z_n \} \langle s \rangle \text{ end, CE} \), push
    \( (\langle s \rangle, CE + \{ z_1 \rightarrow E(\langle y_1 \rangle) \ldots z_n \rightarrow E(\langle y_n \rangle) \} ) \)
    on the stack
- If the activation condition \( E(\langle x \rangle) \) is determined) is false:
  - Suspend
Execution examples

```plaintext
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 ([(\(s\)\(_1\), \(\emptyset\)], \(\emptyset\))

After local Max C in ...
( [(\(s\)\(_2\), \{Max \(\rightarrow m\), C \(\rightarrow c\}\)], \{m, c\} )

After Max binding
( [(\(s\)\(_4\), \{Max \(\rightarrow m\), C \(\rightarrow c\}\}],
\{m = (proc\{\(\$\) X Y Z\} \(s\)\(_3\) end, \(\emptyset\), c\} )

Execution examples (3)

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

• After Max binding

( [((s)_4, {Max → m, C → c})],
  {m = (proc{$ X Y Z$} (s)_3 end , ∅) , c} )

• After procedure call

( [((s)_3, {X → t_1, Y → t_2, Z → c}) ],
  {m = (proc{$ X Y Z$} (s)_3 end , ∅) , t_1=3, t_2=5, c} )
Execution examples (4)

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

〈s〉1
{〈s〉2
  {〈s〉3
    if X >= Y then Z=X else Z=Y end
  }
  {〈s〉4
    {Max 3 5 C}
  }
}

• After procedure call
  ( [(〈s〉3, {X → t₁, Y → t₂, Z → c})],
   \(\{m = (\text{proc}\{\$ X Y Z\} \langle s \rangle_3 \text{ end} , \emptyset \}) , t_1=3, t_2=5, c\} )

• After T = (X>=Y)
  ( [(〈s〉3, {X → t₁, Y → t₂, Z → c, T → t})] ,
   \(\{m = (\text{proc}\{\$ X Y Z\} \langle s \rangle_3 \text{ end} , \emptyset \}) , t_1=3, t_2=5, c, t=false\} )

• ( [(Z=Y , {X → t₁, Y → t₂, Z → c, T → t})] ,
   \(\{m = (\text{proc}\{\$ X Y Z\} \langle s \rangle_3 \text{ end} , \emptyset \}) , t_1=3, t_2=5, c, t=false\} )
Execution examples (5)

\[
\begin{align*}
\langle s \rangle_1 & \left\{ \langle s \rangle_2 \left\{ \langle s \rangle_3 \left\{ \text{local Max C in} \right. \right. \\
& \left. \left. \text{proc \{Max X Y Z\}} \right. \right. \\
& \left. \left. \text{if X >= Y then Z=X else Z=Y end} \right. \right. \\
& \left. \langle s \rangle_4 \{\text{Max 3 5 C}\} \right. \right. \\
& \left. \text{end} \right. \right. \\
\right. \right. \\
\end{align*}
\]

- \(( [(Z=Y, \{X \rightarrow t_1, Y \rightarrow t_2, Z \rightarrow c, T \rightarrow t}})] )
  \langle s \rangle_3 \text{end , } \emptyset, \langle s \rangle_4 \{\text{Max 3 5 C}\} \end{align*}
\]

- \(\emptyset\)
Procedures with external references

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

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

- 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

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

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

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

• STACK: \[\left(\{\text{LB T C}\}, \{Y \rightarrow y, \text{LB } \rightarrow lb, \text{C } \rightarrow c, \text{T } \rightarrow t\}\right)\]

• STORE: \[\{y = 5, lb = (\text{proc}\{\text{$X Z$}\} \langle s \rangle_3 \text{end}, \{Y \rightarrow y\})\}, t = 3, c\}\]

• STACK: \[\left(\langle s \rangle_3, \{Y \rightarrow y, X \rightarrow t, Z \rightarrow c\}\right)\]

• STORE: \[\{y = 5, lb = (\text{proc}\{\text{$X Z$}\} \langle s \rangle_3 \text{end}, \{Y \rightarrow y\})\}, t = 3, c\}\]
Procedures with external references

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

- STACK: \[\langle s \rangle_3, \{Y \rightarrow y, X \rightarrow t, Z \rightarrow c\}\] ]
- STORE: \(\{y = 5, lb = (\text{proc}\{X Z\} \langle s \rangle_3 \text{ end }, \{Y \rightarrow y\}), t = 3, c\}\)
- STACK: \[(Z=Y, \{Y \rightarrow y, X \rightarrow t, Z \rightarrow c\})\] ]
- STORE: \(\{y = 5, lb = (\text{proc}\{X Z\} \langle s \rangle_3 \text{ end }, \{Y \rightarrow y\}), t = 3, c\}\)
- STACK: [ ]
- STORE: \(\{y = 5, lb = (\text{proc}\{X Z\} \langle s \rangle_3 \text{ end }, \{Y \rightarrow y\}), t = 3, c = 5\}\)
From the kernel language to a practical language

- **Interactive interface**
  - the `declare` statement and the global environment

- **Extend kernel syntax** to give a full, practical syntax
  - nesting of partial values
  - implicit variable initialization
  - expressions
  - nesting the `if` and `case` statements
  - `andthen` and `orelse` operations

- **Linguistic abstraction**
  - Functions

- **Exceptions**
The interactive interface (declare)

• The interactive interface is a program that has a single global environment

\textit{declare} X Y

• Augments (and overrides) the environment with new mappings for X and Y

\{Browse X\}

• Inspects the store and shows partial values, and incremental changes
The interactive interface (declare)
declare \( X \ Y \)
Syntactic extensions

• Nested partial values
  – person(name: “George” age:25)
    local A B in A= “George” B=25 person(name:A age:B) end

• Implicit variable initialization
  – local ⟨pattern⟩ = ⟨expression⟩ in ⟨statement⟩ end

• Example:
  assume T has been defined, then
  local tree(key:A left:B right:C value:D) = T in ⟨statement⟩ end
  is the same as:
  local A B C D in
    T = tree(key:A left:B right:C value:D) <statement>
  end
Extracting fields in local statement

```
declare T
    :
    T = tree(key: seif age: 48 profession: professor)
    :
local
    tree(key: A ...) = T
in
    ⟨statement⟩
end
```
Nested if and case statements

- Observe a pair notation is: 1 # 2, is the tuple ‘#’(1 2)

  ```
  case Xs # Ys
  of  nil # Ys then 〈s〉1
  [ Xs # nil then 〈s〉2
  [ (X|Xr) # (Y|Yr) andthen X=<Y then 〈s〉3
  else 〈s〉4 end
  end
  ```

- Is translated into (assuming X,Xr,Y,Yr not free in 〈s〉4)

  ```
  case Xs of nil then 〈s〉1
  else
    case Ys of nil then 〈s〉2
    else
      case Xs of X|Xr then
        case Ys of Y|Yr then
          if X=<Y then 〈s〉3 else 〈s〉4 end
        else 〈s〉4 end
      else 〈s〉4 end
    end
  end
  ```
Expressions

• An expression is a sequence of operations that returns a value
• A statement is a sequence of operations that does not return a value. Its effect is on the store, or outside of the system (e.g. read/write a file)
• $11 \times 11$ $\quad$ $X = 11 \times 11$
  
  expression $\quad$ statement
Functions as linguistic abstraction

- $R = \{F \ X_1 \ldots \ X_n\}$
- $\{F \ X_1 \ldots \ X_n \ R\}$

```
fun \{F \ X_1 \ldots \ X_n\}
  \langle \text{statement} \rangle
  \langle \text{expression} \rangle
end
```

```
proc \{F \ X_1 \ldots \ X_n \ R\}
  \langle \text{statement} \rangle
  R = \langle \text{expression} \rangle
end
```
Nesting in data structures

- $Ys = \{F \times\} \| \{\text{Map } Xr \ F\}$
- Is unnested to:
  - `local Y Yr in`
    ```
    Ys = Y | Yr
    \{F \times Y\}
    \{\text{Map } Xr \ F \ Yr\}
    ```
- The unnesting of the calls occurs after the data structure
Functional nesting

- Nested notations that allows expressions as well as statements
- `local R in
  {F X1 ... Xn R}
  {Q R ...}
end`
- Is written as (equivalent to):
- `{Q {F X1 ... Xn} ...}`
Conditional expressions

\[ R = \text{if } \langle \text{expr}\rangle_1 \text{ then } \langle \text{expr}\rangle_2 \text{ else } \langle \text{expr}\rangle_3 \text{ end} \]

\[ \text{fun } \{\text{Max } X \ Y\} \]
\[ \quad \text{if } X \geq Y \text{ then } X \]
\[ \quad \text{else } Y \text{ end} \]
\[ \text{end} \]

\[ \text{proc } \{\text{Max } X \ Y \ R\} \]
\[ \quad \text{if } X \geq Y \text{ then } X \]
\[ \quad \text{else } Y \text{ end } R = \langle \text{expr}\rangle_2 \]
\[ \text{end} \]

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

\[
\text{fun} \{ \text{Max} \ X \ Y \} \\
\text{if} \ X \geq Y \ \text{then} \ X \\
\text{else} \ Y \ \text{end} \\
\text{end}
\]

\[
\text{proc} \ {\{ \text{Max} \ X \ Y \ R \}} \\
\quad \text{R} = \ ( \ \text{if} \ X \geq Y \ \text{then} \ X \\
\quad \quad \text{else} \ Y \ \text{end} \ ) \\
\text{end}
\]

\[
\text{proc} \ {\{ \text{Max} \ X \ Y \ R \}} \\
\quad \text{if} \ X \geq Y \ \text{then} \ R = X \\
\quad \text{else} \ R = Y \ \text{end} \\
\text{end}
\]
andthen and orelse

\( \langle \text{expr} \rangle_1 \text{ andthen } \langle \text{expr} \rangle_2 \)  

if \( \langle \text{expr} \rangle_1 \) then 
\( \langle \text{expr} \rangle_2 \) 
else false end

\( \langle \text{expr} \rangle_1 \text{ orelse } \langle \text{expr} \rangle_2 \)  

if \( \langle \text{expr} \rangle_1 \) then 
true 
else \( \langle \text{expr} \rangle_2 \) end
Function calls

Observe

{F1 {F2 X} {F3 Y}}

local R1 R2 in
R1 = {F2 X}
R2 = {F3 Y}
{F1 R1 R2}
end

The arguments of a function are evaluated first from left to right.
A complete example

fun \{\text{Map} \ Xs \ F\}
   case \ Xs
       of \ \text{nil} \ then \ \text{nil}
       [] \ X|Xr \ then \ \{F \ X\}\{\text{Map} \ Xr \ F\}
   end
end

proc \{\text{Map} \ Xs \ F \ Ys\}
   case \ Xs
       case \ Xs
           of \ \text{nil} \ then \ Ys \ = \ \text{nil}
           [] \ X|Xr \ then \ Y Yr \ in
               Ys \ = \ Y|Yr
               \{F \ X \ Y\}
               \{\text{Map} \ Xr \ F \ Yr\}
       end
   end
end

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

• How to handle exceptional situations in the program?
• Examples:
  – divide by 0
  – opening a nonexistent file
• Some errors are programming errors
• Some errors are imposed by the external environment
• Exception handling statements allow programs to handle and recover from errors
Exceptions

• The error confinement principle:
  – Define your program as a structured layers of components
  – Errors are visible only internally and a recovery procedure corrects the errors: either errors are not visible at the component boundary or are reported (nicely) to a higher level

• In one operation, exit from arbitrary depth of nested contexts
  – Essential for program structuring; else programs get complicated (use boolean variables everywhere, etc.)
Basic concepts

• A program that encounters an error (exception) should transfer execution to another part, the exception handler and give it a (partial) value that describes the error

• \texttt{try \langle s \rangle_1 \ catch \langle x \rangle \ then \langle s \rangle_2 \ end}

• \texttt{raise \langle x \rangle \ end}

• Introduce an exception marker on the semantic stack

• The execution is equivalent to \langle s \rangle_1 \ if \ it \ executes \ without raising an error

• Otherwise, \langle s \rangle_1 \ is aborted and the stack is popped up to the marker, the error value is transferred through \langle x \rangle, and \langle s \rangle_2 \ is executed
Exceptions (Example)

```plaintext
fun {Eval E}
  if {IsNumber E} then E
  else case E
    of plus(X Y) then {Eval X}+{Eval Y}
    times(X Y) then {Eval X}*{Eval Y}
    else raise illFormedExpression(E) end
  end
end
```

C. Varela; Adapted w/permission from S. Haridi and P. Van Roy
Exceptions (Example)

```plaintext
try
  {Browse {Eval plus(5 6) }}
  {Browse {Eval plus(times(5 5) 6) }}
  {Browse {Eval plus(minus(5 5) 6) }}
catch illFormedExpression(E) then
  {System.showInfo "**** illegal expresion ****" # E}
end
```
Try semantics

- The semantic statement is:
  \[(\text{try } \langle s \rangle_1 \text{ catch } \langle y \rangle \text{ then } \langle s \rangle_2 \text{ end, } E)\]
- Push the semantic statement (catch \langle y \rangle then \langle s \rangle_2 end, E) on ST
- Push \((\langle s \rangle_1, E)\) on ST
- Continue to next execution step
Raise semantics

• The semantic statement is
  \((\text{raise } \langle x \rangle \text{ end, } E)\)
• Pop elements off \(ST\) looking for a \texttt{catch} statement:
  – If a \texttt{catch} statement is found, pop it from the stack
  – If the stack is emptied and no \texttt{catch} is found, then stop execution
    with the error message ”Uncaught exception”
• Let \((\text{catch } \langle y \rangle \text{ then } \langle s \rangle \text{ end, } E_c)\) be the \texttt{catch} statement that is found
• Push \((\langle s \rangle, E_c+\{<y> \rightarrow E(<x>)\})\) on \(ST\)
• Continue to next execution step
Catch semantics

• The semantic statement is
  \((\text{catch } \langle x \rangle \text{ then } \langle s \rangle \text{ end}, E)\)
• Continue to next execution step (like skip)
Full exception syntax

- Exception statements (expressions) with multiple patterns and **finally** clause

- Example:
  ```
  FH = {OpenFile "xxxxx"}
  try
    {ProcessFile FH}
  catch X then
    {System.showInfo "***** Exception when processing *****" # X}
  finally {CloseFile FH} end
  ```
finally syntax

\[
\text{try } \langle s \rangle_1 \text{ finally } \langle s \rangle_2 \text{ end}
\]

is converted to:

\[
\text{try } \langle s \rangle_1 \\
\text{catch } X \text{ then} \\
\quad \langle s \rangle_2 \\
\quad \text{raise } X \text{ end} \\
\text{end} \\
\langle s \rangle_2
\]
Case statement revisited

• 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 \((E(\langle x \rangle) \text{ is determined})\) is true:
  – If \(E(\langle x \rangle)\) is a record, 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 (try local \(\langle x_1 \rangle = \langle x \rangle. \langle f_1 \rangle \ldots \langle x_n \rangle = \langle x \rangle. \langle f_n \rangle \) in \(\langle s_1 \rangle\)\)* end
    catch user(E) then raise E end [] E then \(\langle s_2 \rangle\) end, \(E) on the stack, where \(\langle s_1 \rangle\)\)* is the same as \(\langle s_1 \rangle\) except
      raise E end is changed to raise user(E) end
  – Otherwise, push \((\langle s_2 \rangle, E)\) on the stack

• If the activation condition \((E(\langle x \rangle) \text{ is determined})\) is false:
  – Suspend
Exercises

49. VRH Exercise 2.9.3 (page 107).
50. VRH 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.
52. VRH Exercise 2.9.12 (page 110).
53. Change the semantics of the case statement, so that patterns can contain variable labels and variable feature names.
54. Restrict the kernel language to make it strictly functional (i.e., without dataflow variables)
   - Language similar to Scheme (dynamically typed functional language)

   This is done by disallowing variable declaration (without initialization) and disallowing procedural syntax
   - Only use implicit variable initialization
   - Only use functions