

## 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  
October 15, 2007

Adapted with permission from:  
Seif Haridi  
KTH  
Peter Van Roy  
UCL

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

1

## Sequential declarative computation model

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

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

2

## Conditional

- The semantic statement is  
( if  $\langle x \rangle$  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 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

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

3

## Case statement

- The semantic statement is  
( case  $\langle x \rangle$  of  $\langle l \rangle$  ( $\langle f_1 \rangle : \langle x_1 \rangle \dots \langle f_n \rangle : \langle x_n \rangle$ )  
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, the label of  $E(\langle x \rangle)$  is  $\langle l \rangle$  and its arity is [ $\langle f_1 \rangle \dots \langle f_n \rangle$ ]:  
push (local  $\langle x_1 \rangle = \langle x \rangle$ .  $\langle f_1 \rangle \dots \langle x_n \rangle = \langle x \rangle$ .  $\langle f_n \rangle$  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

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

4

## Procedure values

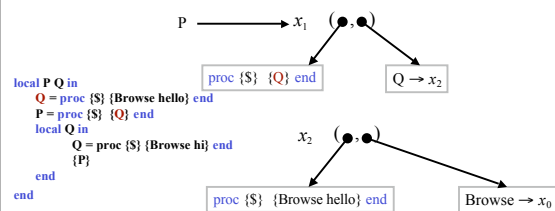
- Constructing a procedure value in the store is not simple because a procedure may have external references

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

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

5

## Procedure values (2)



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

6

## Procedure values (3)

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

```
(proc { $ \langle y_1 \rangle ... \langle y_n \rangle }
  \langle s \rangle
end ,
CE)
```

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

7

## Procedure introduction

- The semantic statement is  
 $((x) = \text{proc } \{ \$ \langle y_1 \rangle \dots \langle y_n \rangle \} \langle s \rangle \text{ end}, E)$
- Create a contextual environment:  
 $CE = E \upharpoonright_{\{\epsilon_j, \dots, \epsilon_k\}}$  where  $\langle z_1 \rangle \dots \langle z_k \rangle$  are external references in  $\langle s \rangle$ .
- Create a new procedure value of the form:  
 $(\text{proc } \{ \$ \langle y_1 \rangle \dots \langle y_n \rangle \} \langle s \rangle \text{ end}, CE)$ , refer to it by the variable  $x_p$
- Bind the store variable  $E(\langle x \rangle)$  to  $x_p$
- Continue to next execution step

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

8

## Procedure application

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

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

9

## Execution examples

```
(s)1 { (s)2 { local Max C in
  proc {Max X Y Z}
    (s)3 if X >= Y then Z=X else Z=Y end
  end
  {Max 3 5 C}
end
```

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

10

## Execution examples (2)

```
(s)1 { (s)2 { local Max C in
  proc {Max X Y Z}
    (s)3 if X >= Y then Z=X else Z=Y end
  end
  (s)4 {Max 3 5 C}
end
```

- Initial state  $([(\langle s \rangle_1, \emptyset)], \emptyset)$
- After local Max C in ...  
 $([(\langle s \rangle_2, \{ \text{Max} \rightarrow m, C \rightarrow c \}], \{ m, c \})$
- After Max binding  
 $([(\langle s \rangle_4, \{ \text{Max} \rightarrow m, C \rightarrow c \}], \{ m = (\text{proc } \{ \$ X Y Z \} \langle s \rangle_3 \text{ end}, \emptyset), c \})$

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

11

## Execution examples (3)

```
(s)1 { (s)2 { local Max C in
  proc {Max X Y Z}
    (s)3 if X >= Y then Z=X else Z=Y end
  end
  (s)4 {Max 3 5 C}
end
```

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

12

## Execution examples (4)

$$\langle s \rangle_1 \left\{ \langle s \rangle_2 \left\{ \begin{array}{l} \text{local Max C in} \\ \text{proc } \{\text{Max X Y Z}\} \\ \langle s \rangle_3 \text{ if } X \geq Y \text{ then } Z=X \text{ else } Z=Y \text{ end} \\ \text{end} \\ \langle s \rangle_4 \{\text{Max 3 5 C}\} \\ \text{end} \end{array} \right. \right.$$

- After procedure call  
 $([(\langle s \rangle_3, \{X \rightarrow t_1, Y \rightarrow t_2, Z \rightarrow c\}), \\ m = (\text{proc}\{\$ X Y Z\} \langle s \rangle_3 \text{end}, \emptyset), t_1=3, t_2=5, c] )$
- After  $T = (X \geq Y)$   
 $([(\langle s \rangle_2, \{X \rightarrow t_1, Y \rightarrow t_2, Z \rightarrow c, T \rightarrow t\}), \\ m = (\text{proc}\{\$ X Y Z\} \langle s \rangle_3 \text{end}, \emptyset), t_1=3, t_2=5, c, t=\text{false}] )$
- $([(Z=Y, \{X \rightarrow t_1, Y \rightarrow t_2, Z \rightarrow c, T \rightarrow t\}), \\ m = (\text{proc}\{\$ X Y Z\} \langle s \rangle_3 \text{end}, \emptyset), t_1=3, t_2=5, c, t=\text{false}] )$

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

13

## Execution examples (5)

$$\langle s \rangle_1 \left\{ \langle s \rangle_2 \left\{ \begin{array}{l} \text{local Max C in} \\ \text{proc } \{\text{Max X Y Z}\} \\ \langle s \rangle_3 \text{ if } X \geq Y \text{ then } Z=X \text{ else } Z=Y \text{ end} \\ \text{end} \\ \langle s \rangle_4 \{\text{Max 3 5 C}\} \\ \text{end} \end{array} \right. \right.$$

- $([(Z=Y, \{X \rightarrow t_1, Y \rightarrow t_2, Z \rightarrow c, T \rightarrow t\}), \\ m = (\text{proc}\{\$ X Y Z\} \langle s \rangle_3 \text{end}, \emptyset), t_1=3, t_2=5, c, t=\text{false}] )$
- $([ , \\ m = (\text{proc}\{\$ X Y Z\} \langle s \rangle_3 \text{end}, \emptyset), t_1=3, t_2=5, c=5, t=\text{false}] )$

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

14

## Procedures with external references

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

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

15

## Procedures with external references

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

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

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

16

## Procedures with external references

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

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

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

17

## Procedures with external references

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

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

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

18

## Procedures with external references

```

(s)1 { (s)2 {
  local LB Y C in
  Y = 5
  proc {LB X Z}
  (s)3 { if X >= Y then Z=X else Z=Y end
  end
  {LB 3 C}
  end
}

```

- STACK: [((s)<sub>3</sub>, {Y → y, X → t, Z → c}) ]
- STORE: {y = 5, lb = (proc{\$ X Z} (s)<sub>3</sub> end, {Y → y}), t = 3, c}
- STACK: [(Z=Y, {Y → y, X → t, Z → c}) ]
- STORE: {y = 5, lb = (proc{\$ X Z} (s)<sub>3</sub> end, {Y → y}), t = 3, c}
- STACK: [ ]
- STORE: {y = 5, lb = (proc{\$ X Z} (s)<sub>3</sub> end, {Y → y}), t = 3, c = 5}

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

19

## 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**

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

20

## The interactive interface (declare)

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

`declare X Y`

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

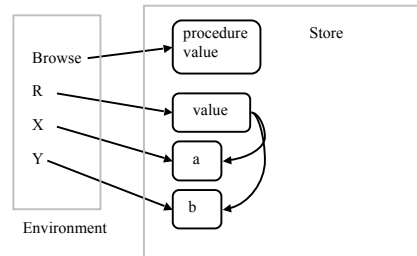
`{Browse X}`

- Inspects the store and shows partial values, and incremental changes

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

21

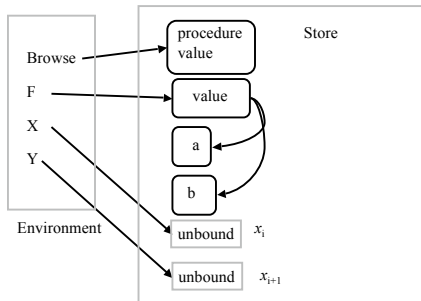
## The interactive interface (declare)



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

22

## declare X Y



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

23

## 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`

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

24

## Extracting fields in local statement

```

declare T
:
T = tree(key:seif age:48 profession:professor)
:
local
tree(key:A ...) = T
in
(statement)
end
    
```

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

25

## 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) and then X=<Y then (s)3
else (s)4 end
    
```

- Is translated into

```

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
    
```

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

26

## 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*11           X=11*11
  { }           { }
expression     statement
    
```

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

27

## Functions as linguistic abstraction

- $R = \{F X1 \dots Xn\}$
- $\{F X1 \dots Xn R\}$

```

fun {F X1 ... Xn}
(statement)
(expression)
end
(statement)
    
```

→

```

proc {F X1 ... Xn R}
(statement)
R = (expression)
end
(statement)
    
```

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

28

## Nesting in data structures

- $Ys = \{F X\}\{\text{Map } Xr F\}$
  - Is unnested to:
- ```

local Y Yr in
Ys = Y|Yr
{F X Y}
{Map Xr F Yr}
end
    
```
- The unnesting of the calls occurs after the data structure

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

29

## 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} ...}
  { }
  expression
  { }
  statement
    
```

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

30

### Conditional expressions

```
R = if (expr)1 then
      (expr)2
    else (expr)3 end
```

(expression)

→

```
if (expr)1 then
  R = (expr)2
else R = (expr)3 end
```

(statement)

```
fun {Max X Y}
  if X>=Y then X
  else Y end
end
```

```
proc {Max X Y R}
  R = ( if X>=Y then X
        else Y end )
end
```

C. Varela; Adapted w/permission from S. Haridi and P. Van Roy 31

### Example

```
fun {Max X Y}
  if X>=Y then X
  else Y end
end
```

→

```
proc {Max X Y R}
  R = ( if X>=Y then X
        else Y end )
end
```

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

C. Varela; Adapted w/permission from S. Haridi and P. Van Roy 32

### andthen and orelse

```
(expr)1 andthen (expr)2
```

→

```
if (expr)1 then
  (expr)2
else false end
```

```
(expr)1 orelse (expr)2
```

→

```
if (expr)1 then
  true
else (expr)2 end
```

C. Varela; Adapted w/permission from S. Haridi and P. Van Roy 33

### 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

C. Varela; Adapted w/permission from S. Haridi and P. Van Roy 34

### A complete example

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

→

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

C. Varela; Adapted w/permission from S. Haridi and P. Van Roy 35

### 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

C. Varela; Adapted w/permission from S. Haridi and P. Van Roy 36

## 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.)

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

37

## 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
  - `try ⟨s⟩1 catch ⟨x⟩ then ⟨s⟩2 end`
  - `raise ⟨x⟩ end`
- Introduce an exception marker on the semantic stack
- The execution is equivalent to ⟨s⟩<sub>1</sub> if it executes without raising an error
- Otherwise, ⟨s⟩<sub>1</sub> is aborted and the stack is popped up to the marker, the error value is transferred through ⟨x⟩, and ⟨s⟩<sub>2</sub> is executed

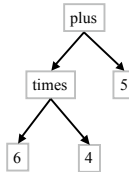
C. Varela; Adapted w/permission from S. Haridi and P. Van Roy

38

## Exceptions (Example)

```

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
end
end
    
```



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

39

## Exceptions (Example)

```

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 expression ****" # E}
end
    
```

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

40

## 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  $(\text{catch } \langle y \rangle \text{ then } \langle s \rangle_2 \text{ end}, E)$  on  $ST$
- Push  $(\langle s \rangle_1, E)$  on  $ST$
- Continue to next execution step

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

41

## Raise semantics

- The semantic statement is  $(\text{raise } \langle x \rangle \text{ end}, E)$
- Pop elements off  $ST$  looking for a `catch` statement:
  - If a `catch` statement is found, pop it from the stack
  - If the stack is emptied and no `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 `catch` statement that is found
- Push  $(\langle s \rangle, E_c + \{ \langle x \rangle \rightarrow E(\langle x \rangle) \})$  on  $ST$
- Continue to next execution step

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

42

## Catch semantics

- The semantic statement is  
(`catch <x> then <s> end, E`)
- Continue to next execution step (like `skip`)

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

43

## 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`

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

44

## Exercises

53. VRH Exercise 2.9.3 (page 107).
54. VRH Exercise 2.9.7 (page 108) –translate example to kernel language and execute using operational semantics.
55. 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.
56. \*VRH Exercise 2.9.12 (page 110).
57. \*Change the semantics of the `case` statement, so that patterns can contain variable labels and variable feature names.

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

45

## Exercises

58. \*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

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

46