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 20, 2009
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
  ( if ⟨x⟩ then ⟨s1⟩ else ⟨s2⟩ end , E)
• If the activation condition (E(⟨x⟩)) is determined) is true:
  – If E(⟨x⟩) is not Boolean (true, false), raise an error
  – E(⟨x⟩) is true, push (⟨s1⟩ , E) on the stack
  – E(⟨x⟩) is false, push (⟨s2⟩ , E) on the stack
• If the activation condition (E(⟨x⟩)) is determined) is false:
  – Suspend

Case statement

• The semantic statement is
  ( case ⟨x⟩ of ⟨l⟩ ⟨f1⟩ : ⟨x1⟩ … ⟨f1⟩ : ⟨x1⟩)
  then ⟨s1⟩
  else ⟨s2⟩ end , E)
• If the activation condition (E(⟨x⟩)) is determined) is true:
  – If E(⟨x⟩) is a record, the label of E(⟨x⟩) is ⟨l⟩ and
  its arity is ⟨⟨f1⟩ … ⟨f1⟩⟩:
    push (local ⟨x1⟩ = ⟨x1⟩. ⟨f1⟩ … ⟨x1⟩ = ⟨x1⟩. ⟨f1⟩ in ⟨s1⟩ end , E)
on the stack
  – Otherwise, push ((⟨s2⟩ , E) on the stack
• If the activation condition (E(⟨x⟩)) is determined) is false:
  – Suspend

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
end P

Procedure values (2)
### Procedure values (3)

- The semantic statement is $(\sigma) = \text{proc } (\lambda y_1 ... y_n ) (E) \text{ end, } CE$
- $(\gamma_1) ... (\gamma_n)$ are the (formal) parameters of the procedure
- Other free identifiers in $(\sigma)$ are called external references $(\alpha_1) ... (\alpha_k)$
- These are defined by the environment $E$ where the procedure is declared (lexical scoping)
- The contextual environment of the procedure $CE$ is $E_{\alpha_1, ..., \alpha_k}$
- When the procedure is called $CE$ is used to construct the environment for execution of $(\sigma)$

### Procedure introduction

- The semantic statement is $(\sigma) = \text{proc } (\lambda y_1 ... y_n ) (E) \text{ end, } CE$
- Create a contextual environment: $CE = E_{\alpha_1, ..., \alpha_k}$ where $(\alpha_1) ... (\alpha_k)$ are external references in $(\sigma)$
- Create a new procedure value of the form: $(\text{proc } (\lambda y_1 ... y_n ) (E) \text{ end, } CE)$, refer to it by the variable $s_r$
- Bind the store variable $E(s)$ to $s_r$
- Continue to next execution step

### Procedure application

- The semantic statement is $(\{ (\sigma) (y_1) ... (y_n ) \}, E)$
- If the activation condition $(E(\sigma))$ is determined
  - If $(E(\sigma))$ is not a procedure value, or it is a procedure with arity that is not equal to $n$, raise an error
  - If $(E(\sigma))$ is $(\text{proc } (\lambda s) ... (\lambda s) (E) \text{ end, } CE)$, push $(\sigma), CE + \{(s_1) \rightarrow E(\gamma_1) ... (s_n) \rightarrow E(\gamma_n)\}$ on the stack
- If the activation condition $(E(\sigma))$ is determined is false:
  - Suspend

### Execution examples (2)

- Initial state $(\{(\sigma)_0, (\emptyset)\}, (\emptyset))$
- After local $\text{Max C}$ in ...
- After $\text{Max binding}$
  - $(\{(\sigma)_3, (\{(\text{Max } m, C \rightarrow c)\}, (m, c)\})$
- After procedure call

### Execution examples (3)

- After $\text{Max binding}$
  - $(\{(\sigma)_4, (\{(\text{Max } m, C \rightarrow c)\}, (m = (\text{proc } (S X Y Z) (\{s_3\} \text{ end, } \emptyset), c)\}$
- After procedure call
  - $(\{(\sigma)_5, (\{(\text{Max } m, C \rightarrow c)\}, (m = (\text{proc } (S X Y Z) (\{s_3\} \text{ end, } \emptyset), (t_1, t_2, t_3, c)\}$

---

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

---

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

\[
\begin{align*}
\text{local Max C in} & \quad \text{proc } [\text{Max X Y Z}] \\
(\delta_1) & \quad (\delta_2) \\
& \begin{cases}
(\delta_1) & \text{if } X \geq Y \text{ then } Z=X \text{ else } Z=Y \text{ end} \\
(\delta_2) & \text{LB 3 C} \\
& \text{end}
\end{cases}
\end{align*}
\]

- After procedure call
  - \( [[(X \rightarrow Y, Y \rightarrow i, Z \rightarrow c)] \rightarrow \text{LB 1 C}] \)
  - \( (m = \text{proc } \{X \text{ Y Z}\} (\delta_1) \text{ end}, \emptyset), t_1 = 3, t_2 = 5, t = \text{false} \)

**Procedures with external references**

\[
\begin{align*}
\text{local LB Y C in} & \quad \text{proc } [\text{LB X Z}] \\
(\delta_1) & \quad (\delta_2) \\
& \begin{cases}
(\delta_1) & \text{if } X \geq Y \text{ then } Z=X \text{ else } Z=Y \text{ end} \\
(\delta_2) & \text{LB 3 C} \\
& \text{end}
\end{cases}
\end{align*}
\]

**Execution examples (5)**

\[
\begin{align*}
\text{local Max C in} & \quad \text{proc } [\text{Max X Y Z}] \\
(\delta_1) & \quad (\delta_2) \\
& \begin{cases}
(\delta_1) & \text{if } X \geq Y \text{ then } Z=X \text{ else } Z=Y \text{ end} \\
(\delta_2) & \text{LB 3 C} \\
& \text{end}
\end{cases}
\end{align*}
\]

- After procedure call
  - \( [[(Y \rightarrow c, X \rightarrow t, Y \rightarrow t, Z \rightarrow c, T \rightarrow t)] \rightarrow \text{LB 1 C}] \)
  - \( (m = \text{proc } \{X \text{ Y Z}\} (\delta_1) \text{ end } \emptyset), t_1 = 3, t_2 = 5, t = \text{false} \)

**Procedures with external references**

\[
\begin{align*}
\text{local LB Y C in} & \quad \text{proc } [\text{LB X Z}] \\
(\delta_1) & \quad (\delta_2) \\
& \begin{cases}
(\delta_1) & \text{if } X \geq Y \text{ then } Z=X \text{ else } Z=Y \text{ end} \\
(\delta_2) & \text{LB 3 C} \\
& \text{end}
\end{cases}
\end{align*}
\]

- The procedure value of LB is
  - \( \text{proc } \{X \text{ Z}\} (\delta_1) \text{ end } \{Y \rightarrow y\} \)
- The store is \( \{y = 5, \ldots\} \)
- STACK: \( \{\text{LB T C} \}, \{Y \rightarrow y, \text{ LB } \rightarrow \text{ lb}, C \rightarrow c, T \rightarrow t\} \}
- STORE: \( \{y = 5, \text{ lb } \rightarrow \text{ proc } \{X \text{ Z}\} (\delta_1) \text{ end }, \{Y \rightarrow y\}, t_3 = 3, c\} \)
- STACK: \( \{\text{ LB T C} \}, \{Y \rightarrow y, X \rightarrow t, Z \rightarrow c\} \}
- STORE: \( \{y = 5, \text{ lb } \rightarrow \text{ proc } \{X \text{ Z}\} (\delta_1) \text{ end }, \{Y \rightarrow y\}, t_3 = 3, c\} \)

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

C. Varela, Adapted w/permission from S. Haridi and P. Van Roy
Procedures with external references

\[ \begin{align*}
\text{local} & \quad B \quad Y \quad C \\
\text{proc} & \quad \begin{cases} 
\lbrace LB \quad X \quad Z \rbrace \\
\text{if} \quad X \geq Y \quad \text{then} \quad Z = X \\
\text{else} \quad Z = Y \\
\text{end} \\
\rbrace \\
\text{end}
\end{cases}
\]

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
  - and then and or else operations
- Linguistic abstraction
  - Functions
  - Exceptions

The interactive interface (declare)

- The interactive interface is a program that has a single global environment
  
  \[ \text{declare} \quad X \quad Y \]

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

  \[ \lbrace \text{Browse} \ X \rbrace \]

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

Syntactic extensions

- Nested partial values
  - person(name: “George” age: 25)
  
  \[ \text{local} \quad \text{A} \quad \text{B} \quad \text{in} \quad \text{A} = \text{“George”} \quad \text{B} = 25 \quad \text{person(name:A age:B)} \quad \text{end} \]

- Implicit variable initialization
  - local (pattern) = (expression) in (statement) end

- Example:
  assume T has been defined, then
  
  \[ \text{local} \quad \text{tree(key:A left:B right:C value:D)} = \text{T} \quad \text{in} \quad \text{statement} \quad \text{end} \]

  is the same as:
  
  \[ \text{local} \quad \text{A B C D in} \quad \text{T} = \text{tree(key:A left:B right:C value:D)} \quad \text{<statement>} \quad \text{end} \]
Extracting fields in local statement

```java
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: \( T \# 2 \), is the tuple \( (1, 2) \)
- \( \text{case } Xs \# Ys \text{ of } \) \( \text{if } Ys \text{ then } (s)_1 \)
- \( 	ext{if } Xs \text{ then } (s)_2 \)
- \( (X|Y) \text{ if } Y|Y \text{ then } (s)_3 \)
- \( \text{else } (s)_4 \text{ end} \)

- Is translated into:
  - \( \text{case } Xs \text{ of } \)
  - \( \text{if } Ys \text{ then } (s)_1 \)
  - \( \text{else } (s)_2 \text{ end} \)
  - \( \text{case } Xs \text{ of } \)
  - \( \text{if } Ys \text{ then } (s)_3 \)
  - \( \text{else } (s)_4 \text{ end} \)
  - \( \text{else } (s)_5 \text{ end} \)

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).

- \( 11^*11 \) is an expression.
- \( X=11^*11 \) is a statement.

Functions as linguistic abstraction

- \( R = \{F \ X1 \ldots Xn\} \)
- \( \{F \ X1 \ldots Xn \ R\} \)

Nesting in data structures

- \( Ys = \{F \ X\}[\text{Map } Xr \ F] \)
- Is unnested to:
  - \( \text{local } Yr \text{ in } \)
  - \( Ys = Y|Yr \)
  - \( \{F \ X \ Y\} \)
  - \( \{\text{Map } Xr \ F \ Yr\} \)
end

- The unnesting of the calls occurs after the data structure.

Functional nesting

- Nested notations that allows expressions as well as statements:
  - \( \text{local } R \in \)
  - \( \{F \ X1 \ldots Xn \ R\} \)
  - \( \{Q \ R \ldots \} \)
end

- Is written as (equivalent to):
  - \( \{Q \{F \ X1 \ldots Xn \} \ldots \} \)
  - expression
  - statement
Conditional expressions

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

Example

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

\[ \text{proc } \{ \text{Max X Y R} \} \]
\[ R = \begin{cases} \text{if } X \geq Y \text{ then } X \text{ else } Y \end{cases} \]

andthen and orelse

\[ \langle \text{expr} \rangle_1 \text{ andthen } \langle \text{expr} \rangle_2 \]

\[ \langle \text{expr} \rangle_1 \text{ orelse } \langle \text{expr} \rangle_2 \]

Function calls

\[ \text{observe} \]
\[ \{ F1 [F2 X] [F3 Y] \} \]

The arguments of a function are evaluated first from left to right

A complete example

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

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

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

  \[
  \text{try} (s_1) \text{catch} (x) \text{then} (s_2) \text{end}
  \]

- Introduce an exception marker on the semantic stack
- The execution is equivalent to \((s_1)\) if it executes without
  raising an error
- Otherwise, \((s_1)\) is aborted and the stack is popped up to the
  marker, the error value is transferred through \((x)\), and \((s_2)\)
  is executed

Exceptions (Example)

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

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 expresion **** # E} end

Try semantics

- The semantic statement is
  \(\text{try} (s_1) \text{catch} (y) \text{then} (s_2) \text{end, } E\)
- Push the semantic statement \((\text{catch} (y) \text{then} (s_2) \text{end, } E)\)
  on \(ST\)
- Push \((s_1)\), \(E\) on \(ST\)
- Continue to next execution step

Raise semantics

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

- The semantic statement is 
  \((\text{catch } (x) \text{ then } (x) \text{ end}, E)\)
- Continue to next execution step (like \textit{skip})

Full exception syntax

- Exception statements (expressions) with multiple patterns and \textit{finally} clause
- Example:
  
  \[
  \begin{aligned}
  \text{FH} &= \{ \text{OpenFile "xxxxx"} \} \\
  \text{try} &\{ \text{ProcessFile FH} \} \\
  \text{catch } X \text{ then} &\{ \text{System.showInfo "***** Exception when processing ***** # X"} \} \\
  \text{finally} &\{ \text{CloseFile FH} \} \end{aligned}
  \]

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 \textit{case} statement, so that patterns can contain variable labels and variable feature names.

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