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

\[
\begin{align*}
(\kappa) & \Rightarrow \text{skip} & \text{empty statement} \\
| & (x) = (y) & \text{variable-variable binding} \\
| & (x) = v & \text{variable-value binding} \\
| & \text{local}(x)\in(s)\text{ end} & \text{sequential composition} \\
| & \text{if}(x)\text{ then } (s_1)\text{ else } (s_2)\text{ end} & \text{conditional} \\
| & \text{case}(x)\text{ of}(\text{pattern})\text{ then } (s_1)\text{ else } (s_2)\text{ end} & \text{pattern matching} \\
| & \text{proc}\{ (y_1), \ldots, (y_n) \}(x) & \text{value expression} \\
(\nu) & \Rightarrow ... & \\
(\text{pattern}) & \Rightarrow ... & \\
\end{align*}
\]

Kernel language syntax

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

\[
(\kappa) \Rightarrow \text{skip} & \quad \text{empty statement} \\
| & (x) = (y) & \text{variable-variable binding} \\
| & (x) = v & \text{variable-value binding} \\
| & \text{local}(x)\in(s)\text{ end} & \text{sequential composition} \\
| & \text{if}(x)\text{ then } (s_1)\text{ else } (s_2)\text{ end} & \text{conditional} \\
| & \text{case}(x)\text{ of}(\text{pattern})\text{ then } (s_1)\text{ else } (s_2)\text{ end} & \text{pattern matching} \\
| & \text{proc}\{ (y_1), \ldots, (y_n) \}(x) & \text{value expression} \\
(\nu) \Rightarrow ... & \\
(\text{pattern}) \Rightarrow ... & \\
\]

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

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. \( \{X \mapsto x_1, Y \mapsto x_2\} \)
- A semantic statement is a pair \( (\theta, E) \) where \( \theta \) is a statement
- \( ST \) is a stack of semantic statements
### skip

- The semantic statement is \((\text{skip}, E)\)
- Continue to next execution step

\[
\begin{array}{c}
\text{ST} \\
\alpha \\
\end{array} + 
\begin{array}{c}
\text{ST} \\
\alpha \\
\end{array}
\]

### 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

\[
\begin{array}{c}
\text{ST} \\
\alpha \\
\end{array} + 
\begin{array}{c}
\text{ST} \\
\alpha \\
\end{array} + 
\begin{array}{c}
\text{ST} \\
\alpha \\
\end{array}
\]

### Variable declaration

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

\[
\begin{array}{c}
\text{ST} \\
\sigma \\
\end{array} + 
\begin{array}{c}
\text{E} \\
\end{array} + 
\begin{array}{c}
\text{ST} \\
\sigma \\
\end{array} + 
\begin{array}{c}
\text{ST} \\
\sigma \\
\end{array} + 
\begin{array}{c}
\text{ST} \\
\sigma \\
\end{array}
\]

### 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.

\[
\begin{align*}
\text{local } P & : Q \to \text{proc } \{ \text{Browse hello} \} \text{ end} \\
P & : \text{proc } \{ Q \} \text{ end} \\
\text{local } Q & : \text{proc } \{ \text{Browse hi} \} \text{ end} \\
& \text{end}
\end{align*}
\]

Procedure values (2)

- Procedure values are pairs:

\[
\begin{align*}
\text{proc } \{ y_1 \} \ldots \{ y_n \} & \text{ end, CE} \\
\text{proc } \{ y_1 \} \ldots \{ y_m \} & \text{ end, CE}
\end{align*}
\]

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

Procedure values (3)

- The semantic statement is:

\[
\{ \delta \} = \text{proc } \{ y_1 \} \ldots \{ y_n \} \text{ end, } E
\]

- Create a contextual environment:

\[
CE = E \{ y_1, \ldots , y_n \}
\]

- Create a new procedure value of the form:

\[
\text{proc } \{ y_1 \} \ldots \{ y_n \} \text{ end, } CE
\]

- Referring to it by the variable \( x_p \)

- Bind the store variable \( E(x_p) \) to \( x_p \)

- Continue to next execution step

Procedure values (4)

- Procedure values are pairs:

\[
\begin{align*}
\text{proc } \{ y_1 \} \ldots \{ y_n \} & \text{ end, CE} \\
\text{proc } \{ y_1 \} \ldots \{ y_m \} & \text{ end, CE}
\end{align*}
\]

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

Procedure introduction

- The semantic statement is:

\[
\{ \delta \} = \text{proc } \{ y_1 \} \ldots \{ y_n \} \text{ end, } E
\]

- Create a contextual environment:

\[
CE = E \{ y_1, \ldots , y_n \}
\]

- Create a new procedure value of the form:

\[
\text{proc } \{ y_1 \} \ldots \{ y_n \} \text{ end, } CE
\]

- Referring to it by the variable \( x_p \)

- Bind the store variable \( E(x_p) \) to \( x_p \)

- Continue to next execution step

Suspendable statements

- The remaining statements require \( x \) to be bound in order to execute

- The activation condition \( E(\{ x \}) \) is determined, is that \( x \) be bound to a number, a record or a procedure value

\[
\begin{align*}
\{ \delta \} & \triangleright x_p \\
\text{if } & (x) \text{ then } (x_1) \text{ else } (x_2) \text{ end condition} \\
\text{case } & (x) \text{ of } \{ \text{pattern} \} \text{ then } (x_1) \\
& \text{else } (x_2) \text{ end pattern matching }
\end{align*}
\]
Life cycle of a thread

- **Running**
  - SF not empty
- **Suspended**
  - A & B / Execute
  - if A & not B
  - not A / Terminate
- **Terminated**

Conditional

- The semantic statement is
  \( \text{if } (x) \text{ then } (s_1) \text{ else } (s_2) \text{ end } , E \)
- If the activation condition \( E((x)) \) is determined is true:
  - If \( E((x)) \) is not Boolean \( \text{true} \), \( \text{false} \), raise an error
  - \( E((x)) \) is true, push \((s_1), E \) on the stack
  - \( E((x)) \) is false, push \((s_2), E \) on the stack
- If the activation condition \( E((x)) \) is determined is false:
  - Suspend

Procedure application

- The semantic statement is
  \( ((x) ; (y_1) \ldots (y_n)) , E \)
- If the activation condition \( E((x)) \) is determined is true:
  - If \( E((x)) \) is not a procedure value, or it is a procedure with arity that is not equal to \( n \), raise an error
  - If \( E((x)) \) is \( (\text{proc } [l] (y_1) \ldots (y_n) \text{ end } , CE) \), push \((x) , CE + (\{x_1 \rightarrow E(y_1) \ldots (x_n \rightarrow E(y_n)) \}) \) on the stack
- If the activation condition \( E((x)) \) is determined is false:
  - Suspend

Case statement

- The semantic statement is
  \( \text{case } (k) \text{ of } (l) \{ (y_1) : (s_1) \ldots (y_n) : (s_n) \} \text{ end } , E \)
- If the activation condition \( E((x)) \) is determined is true:
  - If \( E((x)) \) is a record, and the label of \( E((x)) \) is \( (l) \) and its arity is \( \{y_1 \ldots y_n\} \):
    - push local \( (x_1 = x) , (y_1) \ldots (x_n = x) , (y_n) \) in \((s_x) \text{ end } , E \) on the stack
  - Otherwise, push \((s_2), E \) on the stack
- If the activation condition \( E((x)) \) is determined is false:
  - Suspend

Execution examples

\[
\begin{align*}
\text{local Max C in} & \\
\begin{cases}
\begin{align*}
\text{proc } & (\text{Max } X Y Z) \\
\text{if } & X \gg Y \text{ then } Z=X \text{ else } Z=Y \text{ end }
\end{align*}
\end{cases} & \text{end} \\
\text{Max 3 5 C} & \text{end}
\end{align*}
\]

Execution examples (2)

\[
\begin{align*}
\text{local Max C in} & \\
\begin{cases}
\begin{align*}
\text{proc } & (\text{Max } X Y Z) \\
\text{if } & X \gg Y \text{ then } Z=X \text{ else } Z=Y \text{ end }
\end{align*}
\end{cases} & \text{end} \\
\text{Max 3 5 C} & \text{end}
\end{align*}
\]

- Initial state \( (((\langle 2 \rangle), \emptyset), \emptyset) \)
- After local Max C in
  \[
  \begin{align*}
  ( ((\langle 2 \rangle), & \{\text{Max } m, C \rightarrow c\}) , \{m, c\} ) \\
  \end{align*}
  \]
- After Max binding
  \[
  \begin{align*}
  ( ((\langle 2 \rangle), & \{\text{Max } m, C \rightarrow c\}) , \{m = \text{proc } (\text{Max } X Y Z) \langle 2 \rangle \text{ end } , \emptyset \} , c\}) \\
  \end{align*}
  \]
Execution examples (3)

\[
\begin{array}{l}
\text{local Max C in}\ \text{proc}\ \text{[Max X Y Z]}
\end{array}
\]

\[
\begin{array}{l}
(\delta_0)\ \text{if X := Y then Z := X else Z := Y end}
\end{array}
\]

• After Max binding
  \[
  (\{([s_4], \{\delta_0\})\}, \{m := \text{proc}\{s_4\} X Y Z \{\delta_1\} \text{end}, C \rightarrow c\})
  \]

• After procedure call
  \[
  (\{([s_3], \{\delta_0\})\}, \{m := \text{proc}\{s_3\} X Y Z \{\delta_1\} \text{end}, C \rightarrow c\})
  \]

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

Execution examples (4)

\[
\begin{array}{l}
\text{local Max C in}\ \text{proc}\ \text{[Max X Y Z]}
\end{array}
\]

\[
\begin{array}{l}
(\delta_0)\ \text{if X := Y then Z := X else Z := Y end}
\end{array}
\]

• After procedure call
  \[
  (\{([s_3], \{\delta_0\})\}, \{m := \text{proc}\{s_3\} X Y Z \{\delta_1\} \text{end}, t_1 = 3, t_2 = 5, c\})
  \]

• After T = (X >= Y)
  \[
  (\{([s_3], \{\delta_0\})\}, \{m := \text{proc}\{s_3\} X Y Z \{\delta_1\} \text{end}, t_1 = 3, t_2 = 5, c, t = false\})
  \]

• (\[
  (\{([s_3], \{\delta_0\})\}, \{m := \text{proc}\{s_4\} X Y Z \{\delta_1\} \text{end}, c, t = false\})
  \]

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

Execution examples (5)

\[
\begin{array}{l}
\text{local Max C in}\ \text{proc}\ \text{[Max X Y Z]}
\end{array}
\]

\[
\begin{array}{l}
(\delta_0)\ \text{if X := Y then Z := X else Z := Y end}
\end{array}
\]

\[
\begin{array}{l}
(\delta_0)\ \text{[Max 3 5 C]}
\end{array}
\]

• After procedure call
  \[
  (\{([s_3], \{\delta_0\})\}, \{m := \text{proc}\{s_3\} X Y Z \{\delta_1\} \text{end}, t_1 = 3, t_2 = 5, c, t = false\})
  \]

• (\[
  (\{([s_3], \{\delta_0\})\}, \{m := \text{proc}\{s_4\} X Y Z \{\delta_1\} \text{end}, c, t = false\})
  \]

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

Exercises

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

51. VRH Exercise 2.9.2 (page 107)

52. *After translating the following function to the kernel language:

\[
\text{fun}\ \text{[AddList L1 L2]}
\]

\[
\begin{array}{l}
\text{case}\ \text{L1 of}\ \text{H1|T1}
\end{array}
\]

\[
\begin{array}{l}
\text{case}\ \text{L2 of}\ \text{H2|T2}
\end{array}
\]

\[
\begin{array}{l}
\text{H1+H2|AddList L1 T2)
\end{array}
\]

\[
\begin{array}{l}
\text{nil}\ \text{m and end}
\end{array}
\]

Use the operational semantics to execute the call

\[
\text{[AddList \{1 2\} \{3 4\}]
\]

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