Declarative Computation Model
Kernel language semantics
Basic concepts, the abstract machine (VRH 2.4.1-2.4.2)

Carlos Varela
RPI
October 9, 2009

Declarative Computation Model
Sequential declarative computation model

- The single assignment store
  - declarative (dataflow) variables
  - partial values (variables and values are also called entities)

- The kernel language syntax
- The kernel language semantics
  - The environment: maps textual variable names (variable identifiers) into entities in the store
  - Interpretation (execution) of the kernel language elements (statements) by the use of an abstract machine
  - Abstract machine consists of an execution stack of statements transforming the store

Kernel language syntax

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

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

\[
\langle v \rangle \begin{align*}
\Rightarrow & \quad \text{proc} \{ \langle y_1 \rangle \ldots \langle y_n \rangle \} \langle s \rangle \text{ end} \quad \text{value expression} \\
\end{align*}
\]

Examples

- \text{local } X = 1 \text{ end}
- \text{local } X Y Z \text{ in}
  \begin{align*}
  X &= 5 \\
  Y &= 10 \\
  T &= (X=0) \\
  \text{if } T \text{ then } Z = X \text{ else } Z = Y \text{ end} \\
  \text{(Browse } Z) \\
  \end{align*}

- \text{local } S T \text{ in}
  \begin{align*}
  S &= \text{proc} \{ X Y \} Y = X \times X \text{ end} \\
  \text{(Browse } T) \\
  \end{align*}

Procedure abstraction

- Any statement can be abstracted to a procedure by selecting a number of the “free” variable identifiers and enclosing the statement into a procedure with the identifiers as parameters
- \text{if } X \Rightarrow Y \text{ then } Z = X \text{ else } Z = Y \text{ end}
- Abstracting over all variables
  \text{proc} \{ \langle X Y Z \rangle \}
  \begin{align*}
  \text{if } X = Y \text{ then } Z = X \text{ else } Z = Y \text{ end} \\
  \end{align*}
- Abstracting over X and Z
  \text{proc} \{ \langle \text{LowerBound } X \rangle \}
  \begin{align*}
  \text{if } X = Y \text{ then } Z = X \text{ else } Z = Y \text{ end} \\
  \end{align*}

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 \((\langle s \rangle, E)\) 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 $\langle ST, \sigma \rangle$.
- $ST$ is a stack of semantic statements.
- A computation is a sequence of execution states $\langle ST_0, \sigma_0 \rangle \rightarrow \langle ST_1, \sigma_1 \rangle \rightarrow \langle ST_2, \sigma_2 \rangle \rightarrow \ldots$

**Semantics**

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

**skip**

- The semantic statement is $\langle \langle \text{skip}, E \rangle \rangle$.
- Continue to next execution step.

**Sequential composition**

- The semantic statement is $\langle \langle s_1 \rangle \langle s_2 \rangle, E \rangle$.
- Push $\langle \langle s_2 \rangle, E \rangle$ and then push $\langle \langle s_1 \rangle, E \rangle$ on $ST$.
- Continue to next execution step.

**Calculating with environments**

- $E$ is mapping from identifiers to entities (both store variables and values) in the store.
- The notation $E(\langle y \rangle)$ retrieves the entity $x$ associated with the identifier $\langle y \rangle$ from the store.
- The notation $E + \{ \langle y \rangle_1 \rightarrow x_1, \langle y \rangle_2 \rightarrow x_2, \ldots, \langle y \rangle_n \rightarrow x_n \}$ denotes a new environment $E'$ constructed from $E$ by adding the mappings $\{ \langle y \rangle_1 \rightarrow x_1, \langle y \rangle_2 \rightarrow x_2, \ldots, \langle y \rangle_n \rightarrow x_n \}$.
- $E'(\langle z \rangle)$ is $x_i$ if $\langle z \rangle$ is equal to $\langle y \rangle_i$, otherwise $E'(\langle z \rangle)$ is equal to $E(\langle z \rangle)$.
- The notation $E|\{\langle y \rangle_1, \langle y \rangle_2, \ldots, \langle y \rangle_n\}$ denotes the projection of $E$ onto the set $\{\langle y \rangle_1, \langle y \rangle_2, \ldots, \langle y \rangle_n\}$, i.e., $E$ restricted to the members of the set.
Calculating with environments (2)

• $E = \{X \rightarrow 1, Y \rightarrow [2, 3], Z \rightarrow x_i\}$
• $E' = E + \{X \rightarrow 2\}$
• $E'(X) = 2, \ E(X) = 1$
• $E|_{\{X,Y\}}$ restricts $E$ to the 'domain' $\{X,Y\}$,
  i.e., it is equal to $\{X \rightarrow 1, Y \rightarrow [2, 3]\}$

Calculating with environments (3)

• local $X$ in $\ E$
  \hspace{1em} $X \rightarrow 1$
local $X$ in $\ E'$
  \hspace{1em} $X \rightarrow 2$
{Browse $X$}
end
{Browse X}
end

Lexical scoping

• Free and bound identifier occurrences
• An identifier occurrence is bound with respect to a
  statement $\langle s \rangle$ if it is in the scope of a declaration inside $\langle s \rangle$
• 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)

• proc $\{P \ X\}$
  \hspace{1em} local $Y$ in $\ Y \rightarrow 1$
  \hspace{1em} {Browse $Y$}
end
  \hspace{1em} Free Occurrences
  \hspace{1em} Bound Occurrences

Lexical scoping (3)

• local $\ Arg1 \ Arg2$ in $\ Arg1 = 111*111$
  \hspace{1em} $Arg2 = 999*999$
  \hspace{1em} $Res = Arg1*Arg2$
end
  \hspace{1em} Free Occurrences
  \hspace{1em} Bound Occurrences

This is not a runnable program!

Lexical scoping (4)

• local $Res$ in $\ Arg1 \ Arg2$ in $\ Arg1 = 111*111$
  \hspace{1em} $Arg2 = 999*999$
  \hspace{1em} $Res = Arg1*Arg2$
end
  \hspace{1em} {Browse Res}
end
Lexical scoping (5)

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

Exercises

46. Translate the following function to the kernel language:

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

47. Translate the following function call to the kernel language:

```
{Browse {Max 5 7}}
```

Exercises

48. Explain the difference between static scoping and dynamic scoping. Give an example program that produces different results with static and dynamic scoping.

49. Think of a reason why static scoping may be preferable to dynamic scoping. Think of a reason why dynamic scoping may be preferable to static scoping.