Tutorial 04, Lectures 4 and 5

The first exercies are partly repetitions.

1 Evaluation with Variables

Suppose that now also variables are allowed in arithmetic expressions, see Tutorial 3:

- A variable is described by a tuple `var(A)` where `A` is an atom giving the *variable name*.

An *environment* is a record that has label `env` and has for each variable name a feature that has an integer value.
How can you evaluate these expressions with respect to an environment? Give a specification and an implementation.
Which property must an environment fulfill such that evaluation actually works?

2 Evaluation with Declaration

An obvious way to extend our arithmetical expressions is by also allowing the declaration of variables with immediately assigning a value computed by an expression. This type of expression is commonly referred to as *let*-expression.
So we have additionally:

- A let-expression is a tuple `let(X Y Z)` where `X` is an atom giving the variable name and `Y` and `Z` are expressions.

The first expression inside a let expression defines the value to be assigned to the variable introduced in the let-expression. The second expression can refer to the variable introduced.
For example,

```
{Eval let(x mult(int(2) int(4)) add(var(x) int(3))) env}
```

should return `11`.
One needs environment adjunction as well here which is implemented by record adjunction: `{AdjoinAt R F X}` takes a record `R`, a feature `F`, and a new field `X` and returns a record which has a feature `F` with the value `X` in addition to all features and fields from `R` but `F`.
For example, `{AdjoinAt a(x:1 y:2) z 7}` returns `a(x:1 y:2 z:7)`, whereas `{AdjoinAt a(x:1 y:2) x 7}` returns `a(x:7 y:2)`.

Note

The following exercises are for rehearsing material presented in the fourth lecture. You can start at the tutorial session and continue at home.

3 Abstract Machine Concepts

Please go through the definitions of the following concepts:

- Statement
- Value expression
- Environment
- Semantic statement
- Semantic stack
- Single-assignment store
- Execution state
- Computation

4 Execution Example

Execute

```
local X in
   X=1
   local X in X=2 end
   X=1
end
```

5 Execution Example

Execute

```
local B in
   if B then skip else skip end
end
```

6 Execution Example

Execute

```
local B in
   B = false
   if B then skip else skip end
end
```

7 Environment Adjunction and Projection

In the following we will use environments

$$\begin{array}{rcl} E_1 & = & \{\mathtt{X} \mapsto x_1, \mathtt{Y} \mapsto x_2\} \\ E_2 & = & \{\mathtt{Y} \mapsto x_3, \mathtt{Z} \mapsto x_4\} \\ E_3 & = & \{\mathtt{X} \mapsto x_5, \mathtt{Z} \mapsto x_6\} \end{array}$$

1. Find all possible i and j with $i, j \in \{1, 2, 3\}$ such that for the environment $E = E_i + E_j$:

$$E(\mathtt{X}) = x_1 \qquad \text{and} \qquad E(\mathtt{Z}) = x_6$$

2. Find all possible i and j with $i, j \in \{1, 2, 3\}$ such that for the environment $E = E_i + E_j$:

$$E(\mathtt{X}) = x_1 \qquad \text{and} \qquad E(\mathtt{Y}) = x_2$$

3. Give the environment $E_3|_{\{\mathtt{X}\}}$.

8 Free and Bound Identifiers

List for each of the following statements the free and bound variable identifiers.

1. `local X in {P X Y} end`
2. `{P X Y} local X in {X P Y} end`
3. `local X in local Y in {X Y Z} end end`
4. `proc {P X} local Y in {Q Z Y} end end`
5. `case X of f(Y) then {P Y} else {Q Y} end`

9 External References

List for each of the following procedure definitions the external references.

1. `proc {P X Y} {Q X Y} end`
2. `proc {P X Y} {P X Y} end`
3. `proc {P X Y} {Q Z U} end`
4. `proc {P X Y} local Z in {Q Z U} end end`
5. `proc {P X Y} local Z in {Q Z Y} end end`

Multiple Variables

In the following examples we will allow ourselves to declare multiple variable identifiers by a single local statement (as in the lecture). If you prefer to be exact, see Assignment 11.

10 Execution Example

Execute

```
local A B C P in
   proc {P X Y Z}
      local B in
         B = (X>Y)
         if B then Z=X else Z=Y end
      end
   end
   A=3
   B=4
   {P A B C}
end
```

What does the procedure `P` compute?

11 Declaring Multiple Variables

This assignment extends the kernel language by a declaration statement that introduces multiple variables simultaneously. That is, `local X Y in skip end` is okay and should have the same effect as `local X in local Y in skip end end`.
The statement under consideration is thus

local $\langle x \rangle_1$ ... $\langle x \rangle_n$ in $\langle s \rangle$ end

Give a rule where the statement to be pushed is $\langle s \rangle$.
sectionSlow and Fast Addition
Take the two procedures `SADD` and `FADD` from Lecture 06. Execute in some more detail the following statements, where you can start from an environment and store that already contain the appropriate identifiers and values for `SADD` and `FADD`.

```
local X Y Z in X=2 Y=3 {SADD X Y Z} end
local X Y Z in X=2 Y=3 {FADD X Y Z} end
```

12 Is Append Tail-Recursive?

Rewrite the following definition of `Append` into kernel language:

```
fun {Append Xs Ys}
   case Xs
   of nil  then Ys
   [] X|Xr then X|{Append Xr Ys}
   end
end
```

Remember that nested value construction is always moved before nested procedure application.

Can you give a reason why nested value construction is given preference over procedure call?

Execute with the abstract machine

```
local Xs Ys Zs in Xs=[1 2] Ys=[3] {Append Xs Ys Zs}
```

where you can again assume that environment and store contain the necessary identifiers and values for `Append`.

Is `Append` tail-recursive? If yes, why? Which role do single-assignment variables play here?