We will run a test program on SAdd and FAdd to see how they differ in their implementation. Both take two parameters and return the sum of the parameters, but they implement the function in different ways.
proc \{SAdd N M R\}

if N==0 then
    R=M \(\text{\textcolor{red}{<s3>}}\)
else
    local N1 N2 in
        N1=N-1 \(\text{\textcolor{red}{<s5>}}\)
        \{SAdd N1 M N2\} \(\text{\textcolor{red}{<s6>}}\)
        R=1+N2 \(\text{\textcolor{red}{<s7>}}\)
end

Cheating! In kernel language, N==0 have to be evaluated first with its boolean result stored in a variable.

Cheating again, in kernel language, this should be converted to two nested locals.

Start off by pushing the whole program on the semantic stack, with empty environment and empty store.

Pop the local, create the four variables x1, y1, z1, p. Create the proper environment and push the body of local on the stack.
Sequential decomposition, split up the statements into two statements with identical environments, don’t touch the store

$p$ in the store is now equal to the procedure AND it’s contextual environment, which consists of the external reference SAAdd

Pop the stack, perform sequential decomposition of the statement into two statements with identical environments

Pop the stack, make the identifier X reference in the store equal to 2

Pop the stack, perform sequential decomposition of the statement into two statements with identical environments
Execution Example 1 slide 4/7

```
(  
  (  Y=3,  
      {X? x1,Y? y1,Z? z1,SAdd? p}  
    ),  
    (  
      {SAdd X Y Z},  
      {X? x1,Y? y1,Z? z1,SAdd? p}  
    )  
  ),  
  { x1=2, y1, z1, p? (proc ($ N M R) <s1> end,{SAdd? p}) }  
)
```

Here starts the actual application

```
(  
  (  <s1>,  
      {SAdd? p, N? x1, M? y1, R? z1}  
    ),  
    { x1=2, y1=3, z1, p? (proc ($ N M R) <s1> end,{SAdd? p}) }  
)
```

```
(  
  (  <s4>,  
      {SAdd? p, N? x1, M? y1, R? z1}  
    ),  
    { x1=2, y1=3, z1, p? (proc ($ N M R) <s1> end,{SAdd? p}) }  
)
```

```
(  
  (  <s8>,  
      {SAdd? p, N? x1, M? y1, R? z1, N1? n1, N2? n2}  
    ),  
    { n1, n2, x1=2, y1=3, z1, p? (proc ($ N M R) <s1> end, {SAdd? p}) }  
)
```

```
(  
  (  <s5>,  
      {SAdd? p, N? x1, M? y1, R? z1, N1? n1, N2? n2}  
    ),  
    (  
      <s9>,  
      {SAdd? p, N? x1, M? y1, R? z1, N1? n1, N2? n2}  
    )  
  )  
  { n1, n2, x1=2, y1=3, z1, p? (proc ($ N M R) <s1> end, {SAdd? p}) }  
)
```

- Pop the stack, perform sequential decomposition of the statement into two statements with identical environments
- Pop the stack, make the identifier Y reference in the store equal to 3
- Push the body of the function p on the stack, its environments consists of its contextual context and mappings for the formal parameters
- Pop the stack, since the condition for the if case is false, push the else statement
- Pop the local statement, push its inner statement and create mappings and store variables
- Sequential decomposition with identical environments
Execution Example 1 slide 5/7

\[
\text{(}\quad\quad(\text{<s9>}, \quad \{\text{SAdd} \ p, N \ x1, M \ y1, R \ z1, N1? \ n1, N2? \ n2}\} ) , \{ \ n1=1, n2, x1=2, y1=3, z1, p? \ (\text{proc} (\$ M R) \text{<s1> end}, \{\text{SAdd}? \ p}) \} )
\]

\[
\text{(}<s6>,\quad \{\text{SAdd}? \ p, N \ x1, M \ y1, R \ z1, N1? \ n1, N2? \ n2}\},
\quad (\text{<s7>}, \quad \{\text{SAdd}? \ p, N \ x1, M \ y1, R \ z1, N1? \ n1, N2? \ n2}\} ) , \{ \ n1=1, n2, x1=2, y1=3, z1, p? \ (\text{proc} (\$ M R) \text{<s1> end}, \{\text{SAdd}? \ p}) \} )
\]

\[
\text{(}<s1>,\quad \{\text{SAdd}? \ p, N \ n1, M \ y1, R \ n2}\},
\quad (\text{<s7>}, \quad \{\text{SAdd}? \ p, N \ x1, M \ y1, R \ z1, N1? \ n1, N2? \ n2}\} ) , \{ \ n1=1, n2, x1=2, y1=3, z1, p? \ (\text{proc} (\$ M R) \text{<s1> end}, \{\text{SAdd}? \ p}) \} )
\]

\[
\text{(}<s4>,\quad \{\text{SAdd}? \ p, N \ n1, M \ y1, R \ n2}\},
\quad (\text{<s7>}, \quad \{\text{SAdd}? \ p, N \ x1, M \ y1, R \ z1, N1? \ n1, N2? \ n2}\} ) , \{ \ n1=1, n2, x1=2, y1=3, z1, p? \ (\text{proc} (\$ M R) \text{<s1> end}, \{\text{SAdd}? \ p}) \} )
\]

\[
\text{(}<s8>,\quad \{\text{SAdd}? \ p, N \ n1, M \ y1, R \ n2, N1? \ n3, N2? \ n4}\}, 
\quad (\text{<s7>}, \quad \{\text{SAdd}? \ p, N \ x1, M \ y1, R \ z1, N1? \ n1, N2? \ n2}\} ) , \{ \ n3, n4, n1=1, n2, x1=2, y1=3, z1, 
\quad p? \ (\text{proc} (\$ M R) \text{<s1> end}, \{\text{SAdd}? \ p}) \} )
\]

Pop the stack, decrement N, and n1 binds to 1
Sequential decomposition
Pop the recursive call, push the body and create the appropriate environment
Notice how <s7> remains on the stack for each recursive call...SLOW!
The condition for the if is false, and thus the else statement is pushed!
Pop the local statement and create the local variables for N1 and N2

ali@sics.se 5
Execution Example 1 slide 6/7

$$\text{( <s5>,}$$
$$\{\text{SAdd? p, N? n1, M? y1, R? n2, N1? n3, N2? n4}\}$$
$$\text{)},$$
$$\text{( <s9>,}$$
$$\{\text{SAdd? p, N? n1, M? y1, R? n2, N1? n3, N2? n4}\}$$
$$\text{)},$$
$$\text{( <s7>,}$$
$$\{\text{SAdd? p, N? x1, M? y1, R? z1, N1? n1, N2? n2}\}$$
$$\text{)},$$
$$\text{,}$$
$$\{\text{n3=0, n4, n1=1, n2, x1=2, y1=3, z1,}$$
$$\text{p? (proc ($ N M R$) <s1> end, \{SAdd? p\}) }\}$$

$$\text{( <s9>,}$$
$$\{\text{SAdd? p, N? n1, M? y1, R? n2, N1? n3, N2? n4}\}$$
$$\text{)},$$
$$\text{( <s7>,}$$
$$\{\text{SAdd? p, N? x1, M? y1, R? z1, N1? n1, N2? n2}\}$$
$$\text{)},$$
$$\text{,}$$
$$\{\text{n3=0, n4, n1=1, n2, x1=2, y1=3, z1,}$$
$$\text{p? (proc ($ N M R$) <s1> end, \{SAdd? p\}) }\}$$

Sequential decomposition

$N1$ is decreased, and thus $n3=0$

Sequential decomposition
Execution Example 1 slide 7/7

Recursive call, stack continues to grow

This time the if conditional is true and thus the first statement is pushed on the stack

Now the inner-most recursion binds its output value, and thus n4=3

The identifier R's referenced value n2 is bound to 4

Last statement is popped and the recursion is done, z1=5, we're finished!!
Execution Example 2 slide 1/7

\[
\text{proc } \{ \text{FAdd N M R} \} \\
\begin{aligned}
\text{if } & N==0 \text{ then} \\
\quad & R=M \\
\text{else} \\
\quad & \text{local } N1 \text{ N2 in} \\
\quad & \quad N1=N-1 \\
\quad & \quad N2=M+1 \\
\quad & \quad \{ \text{FAdd N1 N2 R} \} \\
\end{aligned}
\]

Start off by pushing the whole program on the semantic stack, with an empty environment and an empty store.

Pop the \textit{local}, create the four variables x1, y1, z1, p. Create the proper environment and push the body of \textit{local} on the stack.
Execution Example 2 slide 2/7

([ ( FAdd = proc {FAdd N M R} <s1> end, 
   {X? x1, Y? y1, Z? z1, FAdd? p} 
 ), ( X=2 
   Y=3 
   {FAdd X Y Z} 
   , {X? x1, Y? y1, Z? z1, FAdd? p} 
 ) ] , {x1, y1, z1, p})

Sequential decomposition, split up the statements into two statements with identical environments, don’t touch the store

([ ( X=2 
   Y=3 
   {FAdd X Y Z} 
   , {X? x1, Y? y1, Z? z1, FAdd? p} 
 ) ] , {x1, y1, z1, p? (proc ($ N M R) <s1> end, {FAdd? p})} ) )

p in the store is now equal to the procedure AND it’s contextual environment, which consists of the external reference FAdd

([ ( X=2, 
   {X? x1, Y? y1, Z? z1, FAdd? p} 
 ), ( Y=3 
   {FAdd X Y Z} 
   , {X? x1, Y? y1, Z? z1, FAdd? p} 
 ) ] , {x1, y1, z1, p? (proc ($ N M R) <s1> end, {FAdd? p})} ) )

Pop the stack, perform sequential decomposition of the statement into two statements with identical environments

([ ( Y=3 
   {FAdd X Y Z} 
   , {X? x1, Y? y1, Z? z1, FAdd? p} 
 ) ] , {x1=2, y1, z1, p? (proc ($ N M R) <s1> end, {FAdd? p})} ) )

Pop the stack, make the identifier X reference in the store equal to 2

([ ( Y=3, 
   {X? x1, Y? y1, Z? z1, FAdd? p} 
 ), ( {FAdd X Y Z}, 
   {X? x1, Y? y1, Z? z1, FAdd? p} 
 ) ] , {x1=2, y1, z1, p? (proc ($ N M R) <s1> end, {FAdd? p})} ) )

Pop the stack, perform sequential decomposition of the statement into two statements with identical environments
 Execution Example 2 slide 3/7

( [ [ Y=3, {X? x1,Y? y1,Z? z1,FAdd? p} ] , [ x1=2, y1, z1, p? (proc ($ N M R) <s1> end,{FAdd? p}) } ] )

( [ [ FAdd X Y Z], [X? x1,Y? y1,Z? z1,FAdd? p} ] , [ x1=2, y1=3, z1, p? (proc ($ N M R) <s1> end,{FAdd? p}) } ]

Here starts the actual application

( [ [ <s1>, {FAdd? p, N? x1, M? y1, R? z1} ] , [ x1=2, y1=3, z1, p? (proc ($ N M R) <s1> end,{FAdd? p}) } ]

( [ [ <s4>, {FAdd? p, N? x1, M? y1, R? z1} ] , [ x1=2, y1=3, z1, p? (proc ($ N M R) <s1> end,{FAdd? p}) } ]

( [ [ <s8>, {FAdd? p, N? x1, M? y1, R? z1, N1? n1, N2? n2} ] , [ n1, n2, x1=2, y1=3, z1, p? (proc ($ N M R) <s1> end, {FAdd? p}) } ]

( [ [ <s5>, {FAdd? p, N? x1, M? y1, R? z1, N1? n1, N2? n2} ], [ <s9>, {FAdd? p, N? x1, M? y1, R? z1, N1? n1, N2? n2} ] , [ n1, n2, x1=2, y1=3, z1, p? (proc ($ N M R) <s1> end, {FAdd? p}) } ]

Pop the stack, perform sequential decomposition of the statement into two statements with identical environments

Pop the stack, make the identifier Y's reference in the store equal to 3

Push the body of the function p on the stack, its environments consists of its contextual context and mappings for the formal parameters

Pop the stack, since the condition for the if case is false, push the else statement

Pop the local statement, push its inner statement and create mappings and store variables

Sequential decomposition with identical environments
Execution Example 2 slide 4/7

([ (<s9>,
    {FAdd? p, N? x1, M? y1, R? z1, N1? n1, N2? n2}
  ) ], { n1=1, n2, x1=2, y1=3, z1, p? (proc ($ N M R) <s1> end, {FAdd? p}) } )

Sequential decomposition

([ (<s6>,
    {FAdd? p, N? x1, M? y1, R? z1, N1? n1, N2? n2}
  ) , { n1=1, n2, x1=2, y1=3, z1, p? (proc ($ N M R) <s1> end, {FAdd? p}) } )

Pop the stack, increment M, and bind the value 4 to n2

([ (<s7>,
    {FAdd? p, N? x1, M? y1, R? z1, N1? n1, N2? n2}
  ) ], { n1=1, n2=4, x1=2, y1=3, z1, p? (proc ($ N M R) <s1> end, {FAdd? p}) } )

Pop the local statement and create the local variables for N1 and N2

([ (<s8>,
    {FAdd? p, N? n1, M? n2, R? z1, N1? n3, N2? n4}
  ) ], { n3, n4, n1=1, n2=4, x1=2, y1=3, z1, p? (proc ($ N M R) <s1> end, {FAdd? p}) } )

Pop the stack, decrement N, and bind the value 1 to n1

([ (<s1>,
    {FAdd? p, N? n1, M? n2, R? z1}
  ) ], { n1=1, n2=4, x1=2, y1=3, z1, p? (proc ($ N M R) <s1> end, {FAdd? p}) } )

Pop the recursive call, push the body of p and create the appropriate environment

([ (<s4>,
    {FAdd? p, N? n1, M? n2, R? z1}
  ) ], { n1=1, n2=4, x1=2, y1=3, z1, p? (proc ($ N M R) <s1> end, {FAdd? p}) } )

Pop the stack, increment M, and bind the value 4 to n2

([ (<s7>,
    {FAdd? p, N? x1, M? y1, R? z1, N1? n1, N2? n2}
  ) ], { n1=1, n2=4, x1=2, y1=3, z1, p? (proc ($ N M R) <s1> end, {FAdd? p}) } )

Notice how <s7> will reoccur on the same position in the stack, and thus use constant stack space!

The condition for the if is false, and thus the else statement is pushed!

ali@sics.se
Sequential decomposition with identical environments

Pop the stack, decrement N, and bind it to n1

Sequential decomposition

Pop the stack, increment M, and bind it to n4

Pop the recursive call, push the body and create the appropriate environment

Notice how <s7> still uses constant stack space!
The condition for the if is true, and therefore <s3> is pushed on the stack.

We have learned:

• How to use the abstract machine.

• In particular, how contextual environments are saved for each procedure, so that all external references are enclosed to achieve lexical scoping.

• Recursive functions and procedures, whose last statement in kernel language is a recursion, are tail recursive! They are faster because they use the same amount of stack space. Non-tail-recursive procedures are slow because they use more memory, consequently the operative system and the virtual machine need to reallocate memory for the programs!
fun {Append Xs Ys}
  case Xs
  of nil then Ys
  [] X|Xr then X|{Append Xr Ys}
  end
  end

proc {Append Xs Ys R}
  case Xs
  of nil then
    R=Ys ← s3
  else
    case Xs
    of ’|’(X Xr) then
      local N1 in
      R=’|’(X N1) ← s6
      {Append Xr Ys N1} ← s7
      end
    end
  end
  end
end