Principles of Programming Languages

Lecture 09

Coroutines
Subroutines vs Coroutines

- Subroutine call/return

```
call A
A

call B
B

call B
B'

return

return
```

- Separate activation
Subroutines vs Coroutines (cont.)

- Coroutine resume/resume

- Non-nested lifetimes $\Rightarrow$ abandon stack
- Activation lifetimes potentially unlimited if no “return”
Simple Coroutines

- No recursion
- Only one activation of each coroutine at any time

```plaintext
resume B:  \*a = pc + 2
jrst \@b
(control returns here)
```
Recursive Coroutines

• Initial resume (call) of $x$:
  - create activation for $x$
  - resume execution at entry point

• resume $y$ : suspend current activation
  - Resume which activation of $y$?

• resume $? \equiv \text{return}$
  - anonymous resume
  - “terminated” activation

• Call $\equiv$ create & resume
Recursive Coroutines—the problem

\[
\begin{align*}
\text{proc } x & \{ \\
\text{call } y & \\
\text{resume } z & \\
\text{return} & \\
\} \\
\text{proc } y & \{ \\
\text{call } z & \\
\text{resume } x & \text{ caller of } z? \\
\text{return} & \\
\} \\
\text{proc } z & \{ \\
\text{resume } y & \\
\text{return} & \\
\} \\
\end{align*}
\]
Recursive Coroutines—solutions

- **SIMULA 67**
  - `return ("detach" in Simula 67) in z` resumes "caller of z"

- **SL5**
  - `return` resumes in latest activation that resumed `z`
  - With bindings at activation creation
SL5

- Activations (called “environments”) are first class objects
- \[ p := \text{procedure} \ ( \ . \ . \ . \ ) \ . \ . \ . \ \text{end} \]
  - Creates a procedure (“template”) and assigns it to \( p \)
- \[ e := \text{create} \ p \]
  - Uses template to create activation (including variable storage, continuation point &c.)
- \[ e := e \\ \text{with} \ ( \ . \ . \ . \ ) \]
  - Transmits arguments (through “transmitters” for each argument)
- \[ \text{resume} \ e \]
  - Suspending current activation and resume in \( e \)
  - Suspending becomes latest resumer of \( e \)
- \[ \text{return} \ [\text{to} \ e] \]
  - Suspending current activation
  - Returns control to most recent resumer [to \( e \)]
- No deallocation of ARs—they are garbage collected
SL5 Primitives

- Let $c$ = currently executing AR
- $e := \text{create } p$
  
  $e = \text{allocate}(p)$
  $e.\text{cont} = \text{entrypoint}(p)$
  $e.\text{creator} = c$
  $e.\text{resumer} = c$
  
  $\text{for each } X \text{ nonlocal in } p \text{ do }$
    
    $t = c$
    
    $\text{while } t \neq \text{nil do}$
      
      $\text{if } X \text{ public in } t \text{ then}$
      
      $e.X.\text{lval} = t.X.\text{lval}$
      
      $\text{else } t = t.\text{creator}$
      
      $\text{if } t == \text{nil then } \text{error}(X)$
    
  
}
SL5 Primitives (cont.)

- $e := e \text{ with } (a_1, a_2, \ldots, a_n)$
  
  $e.\text{par}[1] = \text{transmitter}_1(a_1)$
  
  \[
  \ldots
  \]

- $\text{resume } e$

  $c.\text{cont} = \text{resumepoint}$

  // $e.\text{creator}$ untouched

  $e.\text{resumer} = c$

  $c = e$

  $\text{goto } c.\text{cont}$

  resumepoint:
SL5 Primitives (cont.)

- `return`
  
  ```
  c.cont = resumepoint  
c = c.resumer  
goto c.cont
  ```

- `return to e`
  
  ```
  c.cont = resumepoint  
  // no alteration of e.resumer  
c = e  
goto c.cont
  ```
Procedure Call/Return—special case

\( f(a_1, a_2, \ldots, a_n) \Rightarrow \text{resume } (\text{create } f \text{ with } (a_1, a_2, \ldots, a_n)) \)

\( \text{return} \quad \Rightarrow \text{return} \)

- Binding is dynamic

\[
\begin{align*}
e &= \text{allocate}(f) \\
e.\text{cont} &= \text{entrypoint}(f) \\
e.\text{creator} &= c \quad // \text{“access link”} \\
e.\text{resumer} &= c \quad // \text{dynamic link} \\
&\quad // \text{bind nonlocals using creator chain} \\
e.\text{par}[1] &= \text{transmitter}_1(a_1) \\
&\quad \ldots
\end{align*}
\]
Procedure Call/Return (cont.)

```
e.cont = resumepoint
e.resumer = c  // redundant

resumepoint:
  .  .  .
```

```
e.cont = resumepoint  // never used

  c = c.resumer  // follow dl

resumepoint:
  .  .  .
```
SIMULA 67

- Can create class instances (= objects) subordinate to block (AR) in which created
- All objects are “attached” to some AR during execution
- When suspended, AR is “detached”
- \texttt{class p(...); declarations;begin ... end p;}
  - Defines class template with formal parameters
- \texttt{e :- new p(...);}
  - Creates an object (AR) of class \texttt{p: [ref(p) e;]}
  - Transmits arguments
  - \textit{Commences execution} in AR of \texttt{e}
  - AR \texttt{e} is “attached” to the suspended (creating) AR
SIMULA 67 (cont.)

- **detach**;
  - Suspend current activation
  - Resume in AR to which current is “attached”
  - Current AR marked “detached”
  - Approximately a “return”
  - `end ⇒ detach` (blocks detach when exited)

- **call**(`e`)
  - If `e` is detached, mark AR `e` as “attached” to caller (current AR)
  - Suspend caller (current AR)
  - Resume in AR `e`

- **resume**(`e`)
  - If `e` is detached, suspend current AR and resume execution in AR `e`
  - `e` is “attached” to AR to which current AR is “attached”—**resume** passes its attachment to `e`
SIMULA 67 (cont.)
SIMULA 67 Primitives

- Let \( c = \) currently executing AR
- \( e := \textbf{new} \ p(\ldots); \)
  \[ e = \texttt{allocate}(p) \]
  \{ transmit parameters (CBN, CBV in Simula67)\}
  \( e.\text{cont} = \texttt{entrypoint}(p) \)
  \( e.\text{attached} = c \quad // \text{attacher of } e \text{ is } c \)
  \{ using \( c.\text{sl} \) and \( \text{snl}(p) \) and \( c' \text{'}s \text{snl}, \text{calculate} \)
      \( \text{AR in which } p \text{ was defined (created)} \)
      \& put ptr into t} \)
  \( c.\text{sl} = t \)
  \( c.\text{cont} = \texttt{resumepoint} \)
  \( c.\text{attached} = \text{nil} \)
  \( c = e \)
  \( \texttt{goto } c.\text{cont} \)
  \texttt{resumepoint:}
- `detach;`
  
  ```
  c.cont = resumepoint
  if c.attached == nil then error()
  else {
      t = c.attached
      c.attached = nil
      c = t // back to attacher
      goto c.cont
  }
  resumepoint:
  ```
SIMULA67 Primitives (cont.)

- **call(e)** —no parameters
  
  ```
  if e.attached != nil then error()
  e.attached = c // e attached to caller
  c.cont = resumepoint
  c.attached = nil
  c = e
  goto c.cont
  ```

- **resume(e)**
  
  ```
  if e.attached != nil then error()
  e.attached = c.attached // e inherits attacher
  c.cont = resumepoint
  c.attached = nil
  c = e
  goto c.cont
  ```
outer block: \texttt{begin} \hspace{2em} \texttt{class A; \_\_ detach; \_\_;} \\
\hspace{4em} \texttt{ref(A) U,V;} \\
\hspace{4em} U :- \texttt{new A;} \hspace{4em} \texttt{inner block: begin class B; \_\_ detach; \_\_ ;} \\
\hspace{6em} \texttt{ref(B) X;} \\
\hspace{6em} \texttt{ref(A) W;} \\
\hspace{6em} V :- W :- \texttt{new A;} \\
\hspace{6em} X :- \texttt{new B;} \\
\hspace{6em} \ldots \hspace{4em} \texttt{pc} \rightarrow L: \texttt{call(X);} \\
\hspace{8em} \ldots \hspace{4em} \texttt{end inner block;} \\
\hspace{8em} \ldots \hspace{4em} \texttt{call(V);} \\
\hspace{8em} \ldots \hspace{4em} \texttt{end outer block;}
Example: picture at pc →

Block Inner AR

classe B object

Executing in AR for x

Block outer AR

class A object
Example (cont.): Static Links

- Why is static link from $V = W$ to block outer AR?
- $V : - W : - \text{new}(A)$ done in block inner
- Static binding says static environment is where name is declared (space allocated)
- If static link to inner, $y$ resolves to 3!

Block $Inner$ AR

Block $outer$ AR

class A object
Example: 2 coroutines

begin character ch;

    class aatob: ... below ... end aatob;
    class bbtoc: ... below ... end bbtoc;
    ref(aatob) procA;
    ref(bbtoc) procB;
    procA :- new aatob;
    procB :- new bbtoc;
    call(procA);

end
Example (cont.)

class aatob;
    begin
        detach;
        while true do begin
            ch := inchar;
            if ch = 'a' then
                begin ch := inchar;
                    if ch = 'a' then
                        begin ch := 'b'; resume(procB) end
                    else
                        begin character save;
                            save := ch;
                            ch := 'a'; resume(procB)
                            ch := save; resume(procB)
                        end
                end
            else resume(procB)
        end while
    end aatob;
Example (cont.)

class bbtoc;
    begin
        detach;
        while true do begin
            if ch = 'b' then
                begin
                    resume(procA)
                    if ch = 'b' then
                        outchar('c')
                    else
                        begin
                            outchar('b'); outchar(ch)
                        end
                end
            else
                outchar(ch);
        resume(procA)
        end while
    end bbtoc;
Example (cont.)

- **stdin:**
  
  ```
  bbbbb
  aaaaaa
  ababab
  aabbaabb
  xaaaaaxaabbxababxaaaayyy
  bbbbbb
  aaaaaa
  ^D
  ```

- **stdout:**
  
  ```
  ccb
  ca
  ababab
  ccc
  xcxcbxababxbayyy
  ccc
  cb
  ```