Clicking on Delimited Continuations

`http://okmij.org/ftp/packages/caml-shift.tar.gz`
`http://okmij.org/ftp/ML/caml-web.tar.gz`

Continuation Fest 2008
Tokyo, Japan April 13, 2008

FLOLAC 2008
Taipei, Taiwan July 11, 2008

?

Outline

- **Delimited continuations**
  - Delimited evaluation contexts, processes, breakpoints
  - Control operators shift and reset
  - A taste of formalization

Continuations and Web Services

- A simple TTY application
- CGI and the inversion of control
- Interaction and continuations
- Plain CGI scripts and persistent continuations

Web Transactions

- “Please click the Submit button only once”
- A simple blog as a TTY application
- A simple blog as a CGI application with nested transactions

Continuations are the meanings of evaluation contexts

A context is an expression with a hole

```
print( 42 + abs( 2 * 3 ) )
```

Full context	undelimited continuation function $\text{int} \to \infty$
Partial context	delimited continuation function $\text{int} \to \text{int}$, i.e., take absolute value and add 42

Contexts and continuations are present whether we want them or not

Continuations are the meanings of evaluation contexts

A context is an expression with a hole

```
print( 42 + abs( 2 * 3 ) )
```

Full context	undelimited continuation function $\text{int} \to \infty$
Partial context	delimited continuation function $\text{int} \to \text{int}$, i.e., take absolute value and add 42

Contexts and continuations are present whether we want them or not

Continuations are the meanings of evaluation contexts

A context is an expression with a hole

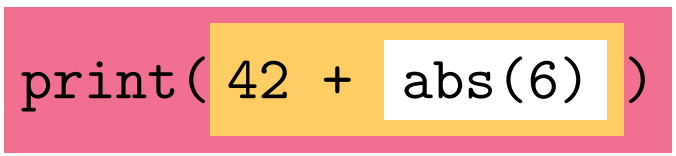
```
print( 42 + abs( 2 * 3 ) )
```

Full context	undelimited continuation function $\text{int} \to \infty$
Partial context	delimited continuation function $\text{int} \to \text{int}$, i.e., take absolute value and add 42

Contexts and continuations are present whether we want them or not

Continuations are the meanings of evaluation contexts

A context is an expression with a hole



Contexts and continuations are present whether we want them or not

Continuations are the meanings of evaluation contexts

A context is an expression with a hole

```
print( 42 + if  6>0  then 6 else neg(6) )
```

Contexts and continuations are present whether we want them or not

Continuations are the meanings of evaluation contexts

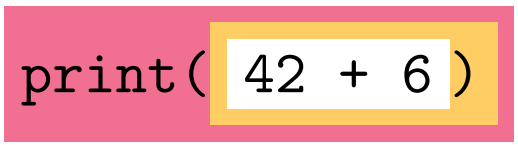
A context is an expression with a hole

```
print( 42 +  if true then 6 else neg(6)  )
```

Contexts and continuations are present whether we want them or not

Continuations are the meanings of evaluation contexts

A context is an expression with a hole



Contexts and continuations are present whether we want them or not

Continuations are the meanings of evaluation contexts

A context is an expression with a hole

Contexts and continuations are present whether we want them or not

Continuations are the meanings of evaluation contexts

A context is an expression with a hole

```
print(48)
```

Contexts and continuations are present whether we want them or not

Control effects: Process scheduling in OS

Operating system, User process, System call

```
schedule( main () {... read(file) ...} ) ...
```

Control effects: Process scheduling in OS

Capture

```
schedule( main () {... read(file) ...} ) ...
```

```
schedule( ReadRequest( PCB ,file) ) ...
```

Control effects: Process scheduling in OS

Capture

```
schedule( main () {... read(file) ...} ) ...
```

```
schedule( ReadRequest( PCB ,file) ) ...
```

...

```
schedule( resume( PCB ,"read string") ) ...
```

Control effects: Process scheduling in OS

Capture, Invoke

```
schedule( main () {... read(file) ...} ) ...

schedule( ReadRequest( PCB ,file) ) ...

...

schedule( resume( PCB ,"read string") ) ...

schedule( main () {... "read string" ...} ) ...
```

Control effects: Process scheduling in OS

Capture

```
schedule( main () {... read(file) ...} ) ...

schedule( ReadRequest( PCB ,file) ) ...

...

schedule( resume( PCB ,"read string") ) ...

schedule( main () {... "read string" ...} ) ...
```

User-level control operations $\Rightarrow$ user-level scheduling, thread library

Control effects as debugging

```
debug_run( 42 + abs(2 * breakpt 1 ) )
```

Control effects as debugging

```
debug_run( 42 + abs(2 * breakpt 1 ) )
```

BP_1

Control effects as debugging

```
debug_run( 42 + abs(2 * breakpt 1 ) )
```

BP_1

```
debug_run(resume (BP1,3))
```

Control effects as debugging

```
debug_run( 42 + abs(2 * breakpt 1 ) )
```

BP_1

```
debug_run(resume (BP1,3))
```

```
debug_run( 42 + abs(2 * 3 ) )
```

Programmable debugger

```
open Delimcc
type breakpt = Done of int | BP of (int -> breakpt)

let p0 = new_prompt ()

let v1 = push_prompt p0 (fun () ->
    Done (42 + abs(2 * shift p0 (fun k -> BP k) ))
```

Programmable debugger

```
open Delimcc
type breakpt = Done of int | BP of (int -> breakpt)

let p0 = new_prompt ()

let v1 = push_prompt p0 (fun () ->
    Done (42 + abs(2 * shift p0 (fun k -> BP k) ))

val v1 : breakpt = BP <fun>
```

Programmable debugger

```
open Delimcc
type breakpt = Done of int | BP of (int -> breakpt)

let p0 = new_prompt ()

let v1 = push_prompt p0 (fun () ->
    Done (42 + abs(2 * shift p0 (fun k -> BP k) ))
  val v1 : breakpt = BP <fun>
let v2 = let BP k = v1 in k 3
```

Programmable debugger

```
open Delimcc      let p0 = new_prompt ()
type breakpt = Done of int | BP of (int -> breakpt)

let v1 = push_prompt p0 (fun () ->
    Done (42 + abs(2 * shift p0 (fun k -> BP k) ))
  val v1 : breakpt = BP <fun>
let v2 = let BP k = v1 in k 3
let v2 = push_prompt p0 (fun () ->
    Done (42 + abs(2 * 3 ))
```

Programmable debugger

```
open Delimcc
type breakpt = Done of int | BP of (int -> breakpt)

let p0 = new_prompt ()

let v1 = push_prompt p0 (fun () ->
    Done (42 + abs(2 * shift p0 (fun k -> BP k) ))
val v1 : breakpt = BP <fun>
let v2 = let BP k = v1 in k 3
let v2 = push_prompt p0 (fun () ->
    Done (42 + abs(2 * 3 ))
val v2 : breakpt = Done 48
```

Programmable debugger

```
open Delimcc
type breakpt = Done of int | BP of (int -> breakpt)

let p0 = new_prompt ()

let v1 = push_prompt p0 (fun () ->
    Done (42 + abs(2 * shift p0 (fun k -> BP k) ))
  val v1 : breakpt = BP <fun>
let v2 = let BP k = v1 in k 3
  val v2 : breakpt = Done 48
let v2' = let BP k = v1 in k (-5)
```

Programmable debugger

```
open Delimcc
type breakpt = Done of int | BP of (int -> breakpt)

let p0 = new_prompt ()

let v1 = push_prompt p0 (fun () ->
    Done (42 + abs(2 * shift p0 (fun k -> BP k) ))
  val v1 : breakpt = BP <fun>
let v2 = let BP k = v1 in k 3
  val v2 : breakpt = Done 48
let v2' = let BP k = v1 in k (-5)
let v2' = push_prompt p0 (fun () ->
    Done (42 + abs(2 * -5 ))
```

Programmable debugger

```
open Delimcc      let p0 = new_prompt ()
type breakpt = Done of int | BP of (int -> breakpt)

let v1 = push_prompt p0 (fun () ->
      Done (42 + abs(2 * shift p0 (fun k -> BP k) ))
  val v1 : breakpt = BP <fun>
let v2 = let BP k = v1 in k 3
  val v2 : breakpt = Done 48
let v2' = let BP k = v1 in k (-5)
let v2' = push_prompt p0 (fun () ->
    Done (42 + abs(2 * -5 ))
  val v2' : breakpt = Done 52
```

Debugging an iteration

```
module CSet =
    Set.Make(struct type t=char let compare=compare end)

let set1 = List.fold_right CSet.add
    ['F';'L';'O';'L';'A';'C';'O';'8']
    CSet.empty
  val set1 : CSet.t = <abstr>

CSet.iter (fun e -> print_char e) set1
  O8ACFLO
```

Debugging an iteration, cont

```
type cursor = EOF | Cons of char * (unit -> cursor)
let pc = new_prompt ()

let sv1 = push_prompt pc (fun () ->
   CSet.iter(fun e-> shift pc (fun k -> Cons (e,k)) ) set1;
   EOF )
   val sv1 : cursor = Cons ('0', <fun>)
```

Debugging an iteration, cont

```
type cursor = EOF | Cons of char * (unit -> cursor)
let pc = new_prompt ()

let sv1 = push_prompt pc (fun () ->
   CSet.iter(fun e-> shift pc (fun k -> Cons (e,k)) ) set1;
   EOF )
   val sv1 : cursor = Cons ('0', <fun>)

let next = function Cons (_,k) -> k ()
let sv2 = next sv1;;
   val sv2 : cursor = Cons ('8', <fun>)
let sv3 = next sv2;;
   val sv3 : cursor = Cons ('A', <fun>)
```

Debugging an iteration, cont

```
type cursor = EOF | Cons of char * (unit -> cursor)
let pc = new_prompt ()

let sv1 = push_prompt pc (fun () ->
   CSet.iter(fun e-> shift pc (fun k -> Cons (e,k)) ) set1;
   EOF )
   val sv1 : cursor = Cons ('0', <fun>)

let rec take n c = match (n,c) with
   | (0,_) | (_,EOF) -> []
   | (n,Cons (e,k)) -> e:: take (pred n) (k ())

take 5 sv2
   - : char list = ['8'; 'A'; 'C'; 'F'; 'L']
```

CBN $\lambda\!\!\!\!\!\!\amalg$-calculus

Primitive Constants $D ::= \text{john} \mid \text{mary} \mid \text{see} \mid \text{tall} \mid \text{mother}$

Constants $C ::= D \mid C \wedge C \mid c \mid \forall_c \mid \partial_c$

Terms $E, F ::= V \mid x \mid FE \mid E \curlywedge F \mid Q \,\$\, E \mid \amalg k : S.\, E$

Values $V ::= C \mid u \mid \lambda x{:}T.\, E \mid W$

Strict Values $W ::= \lambda^{!} u{:}U.\, E$

Coterms $Q ::= \# \mid E, Q \mid Q ;_{!} W \mid E ,_c Q \mid Q ;_c V$

Term equalities

$Q \,\$\, FE = E, Q \,\$\, F \qquad Q \,\$\, WE = Q ;_{!} W \,\$\, E$

$Q \,\$\, F \curlywedge E = E ,_c Q \,\$\, F \qquad Q \,\$\, V \curlywedge E = Q ;_c V \,\$\, E$

$\# \,\$\, V = V$

Transitions

$Q_1 \,\$\, \cdots \,\$\, Q_n \,\$\, (\lambda x.\, E) F \rightsquigarrow Q_1 \,\$\, \cdots \,\$\, Q_n \,\$\, E\{x \mapsto F\}$

$Q_1 \,\$\, \cdots \,\$\, Q_n \,\$\, (\lambda^{!} x.\, E) V \rightsquigarrow Q_1 \,\$\, \cdots \,\$\, Q_n \,\$\, E\{x \mapsto V\}$

$Q_1 \,\$\, \cdots \,\$\, Q_n \,\$\, C_1 \curlywedge C_2 \rightsquigarrow Q_1 \,\$\, \cdots \,\$\, Q_n \,\$\, C_1 \wedge C_2$

$Q_1 \,\$\, \cdots \,\$\, Q_n \,\$\, Q \,\$\, \amalg k.\, E \rightsquigarrow Q_1 \,\$\, \cdots \,\$\, Q_n \,\$\, \# \,\$\, E\{k \mapsto Q\}$

CBN $\lambda\!\!\!\!\!\!\amalg$-calculus

Primitive Constants $D ::= \text{john} \mid \text{mary} \mid \text{see} \mid \text{tall} \mid \text{mother}$

Constants $C ::= D \mid C \wedge C \mid c \mid \forall_c \mid \partial_c$

Terms $E, F ::= V \mid x \mid FE \mid E \curlywedge F \mid Q \,\$\, E \mid \amalg k : S.\, E$

Values $V ::= C \mid u \mid \lambda x{:}T.\, E \mid W$

Strict Values $W ::= \lambda^{!} u{:}U.\, E$

Coterms $Q ::= \# \mid E, Q \mid Q ;_{!} W \mid E ,_c Q \mid Q ;_c V$

Term equalities

$Q \,\$\, FE = E, Q \,\$\, F \qquad Q \,\$\, WE = Q ;_{!} W \,\$\, E$

$Q \,\$\, F \curlywedge E = E ,_c Q \,\$\, F \qquad Q \,\$\, V \curlywedge E = Q ;_c V \,\$\, E$

$\# \,\$\, V = V$

Transitions

$Q_1 \,\$\, \cdots \,\$\, Q_n \,\$\, (\lambda x.\, E)F \rightsquigarrow Q_1 \,\$\, \cdots \,\$\, Q_n \,\$\, E\{x \mapsto F\}$

$Q_1 \,\$\, \cdots \,\$\, Q_n \,\$\, (\lambda^{!} x.\, E)V \rightsquigarrow Q_1 \,\$\, \cdots \,\$\, Q_n \,\$\, E\{x \mapsto V\}$

$Q_1 \,\$\, \cdots \,\$\, Q_n \,\$\, C_1 \curlywedge C_2 \rightsquigarrow Q_1 \,\$\, \cdots \,\$\, Q_n \,\$\, C_1 \wedge C_2$

$Q_1 \,\$\, \cdots \,\$\, Q_n \,\$\, Q \,\$\, \amalg k.\, E \rightsquigarrow Q_1 \,\$\, \cdots \,\$\, Q_n \,\$\, \# \,\$\, E\{k \mapsto Q\}$

Outline

Running example, a console version

Demonstrate `test_Queinnec_tty`, the interactive console version

Running example

```
let main () =
 let henv = inquire "currency_read_rate.html" [] in
 let curr_name = answer "curr-name" henv vstring in
 let curr_rate = answer "rate" henv vfloat in
 let henv = inquire "currency_read_yen.html"
                    [("curr-name",curr_name)] in
 let amount = answer "curr-amount" henv vfloat in
 let yen_amount = amount /. curr_rate in
   inquire_finish "currency_result.html"
   [("curr-name",curr_name);("rate",string_of_float curr_rate);
    ("curr-amount",string_of_float amount);
    ("yen-amount", string_of_float yen_amount)];
   exit 0                                  (* unreachable *)
```

Templates

```
<html><head>
<title>Currency converter with respect to &yen;. Form 2</title>
</head><body>
<H1 ALIGN=CENTER>Example of (delimited) continuations on the Web

<div>${response}</div>

<form action="${this-script}" method="GET">
<input type=hidden name="klabel" value="${klabel}" size=10 maxsi
Converting ${curr-name} into &yen;.
<table>
<tr><td>Enter the amount:
<td align=right>
<input type=text name="curr-amount" value="${curr-amount}" size=
</table>
<INPUT name=submit TYPE=Submit>
</form>
</body></html>
```

Running example

```
let main () =
 let henv = inquire "currency_read_rate.html" [] in
 let curr_name = answer "curr-name" henv vstring in
 let curr_rate = answer "rate" henv vfloat in
 let henv = inquire "currency_read_yen.html"
                    [("curr-name",curr_name)] in
 let amount = answer "curr-amount" henv vfloat in
 let yen_amount = amount /. curr_rate in
   inquire_finish "currency_result.html"
   [("curr-name",curr_name);("rate",string_of_float curr_rate);
    ("curr-amount",string_of_float amount);
    ("yen-amount", string_of_float yen_amount)];
   exit 0                                  (* unreachable *)
```

Running example

```
let main () =
 let henv = inquire "currency_read_rate.html" [] in
 let curr_name = answer "curr-name" henv vstring in
 let curr_rate = answer "rate" henv vfloat in
 let henv = inquire "currency_read_yen.html"
                    [("curr-name",curr_name)] in
 let amount = answer "curr-amount" henv vfloat in
 let yen_amount = amount /. curr_rate in
   inquire_finish "currency_result.html"
   [("curr-name",curr_name);("rate",string_of_float curr_rate);
    ("curr-amount",string_of_float amount);
    ("yen-amount", string_of_float yen_amount)];
   exit 0                                  (* unreachable *)
```

Running example as a typical CGI script

```
let main () =
 let henv = get_form_env () in
 match hlocate "klabel" henv with
 | None -> send "currency_read_rate.html" []
 | Some "got-rate" ->
     (match (hlocate "curr-name" henv, hlocate "rate" henv) wit
     | (Some curr_name, (Some rate as rv)) ->
         let _ = validate rv vfloat in
         send "currency_read_yen.html"
           [("curr-name",curr_name); ("rate",rate)]
     | _ -> failwith "need error handling")
 | Some "got-amount" ->
     (match (hlocate "curr-name" henv, hlocate "rate" henv,
             hlocate "curr-amount" henv) with
     | (Some curr_name, (Some _ as rv), (Some _ as amv)) ->
         let curr_rate = validate rv vfloat in
         let amount = validate amv vfloat in
         let yen_amount = amount /. curr_rate in
         send "currency_result.html" [("curr-name",curr_name);.
     | _ -> failwith "need error handling")
 | _ -> failwith "need error handling";
```

Running example as a typical CGI script

```
let main () =
 let henv = get_form_env () in
 match hlocate "klabel" henv with
 | None -> send "currency_read_rate.html" []
 | Some "got-rate" ->
     (match (hlocate "curr-name" henv, hlocate "rate" henv) wit
     | (Some curr_name, (Some rate as rv)) ->
         let _ = validate rv vfloat in
         send "currency_read_yen.html"
           [("curr-name",curr_name); ("rate",rate)]
     | _ -> failwith "need error handling")
 | Some "got-amount" ->
     (match (hlocate "curr-name" henv, hlocate "rate" henv,
             hlocate "curr-amount" henv) with
     | (Some curr_name, (Some _ as rv), (Some _ as amv)) ->
         let curr_rate = validate rv vfloat in
         let amount = validate amv vfloat in
         let yen_amount = amount /. curr_rate in
         send "currency_result.html" [("curr-name",curr_name);.
     | _ -> failwith "need error handling")
 | _ -> failwith "need error handling";
```

Running example

```
let main () =
 let henv = inquire "currency_read_rate.html" [] in
 let curr_name = answer "curr-name" henv vstring in
 let curr_rate = answer "rate" henv vfloat in
 let henv = inquire "currency_read_yen.html"
                    [("curr-name",curr_name)] in
 let amount = answer "curr-amount" henv vfloat in
 let yen_amount = amount /. curr_rate in
   inquire_finish "currency_result.html"
   [("curr-name",curr_name);("rate",string_of_float curr_rate);
    ("curr-amount",string_of_float amount);
    ("yen-amount", string_of_float yen_amount)];
   exit 0                                  (* unreachable *)
```

Interaction in TTY mode

```
let do_inquire template henv =
  send_form stdout henv template;
  flush_all ();
  let henv = url_unquote_collate (read_line ()) in
  HEnv.add hvar_template_name template henv

 let inquire template outputs =
   let henv =
     List.fold_left (fun m (k,v) -> HEnv.add k v m) HEnv.empty o
   do_inquire template henv

 let inquire_finish template outputs =
   let _ = inquire template outputs in exit 0

 let answer hvar henv validfn =
   validate (fun henv ->
     do_inquire (HEnv.find hvar_template_name henv) henv)
     hvar henv validfn
```

Interaction with Continuations

```
let do_inquire template env =
  let send k =
      let klabel = gensym () in
      Res (Some (klabel,k),template,env) in
  let henv = shift p0 send in
  HEnv.add hvar_template_name template henv

let inquire template outputs =
  do_inquire template (Right outputs)

let inquire_finish template outputs =
    abort p0 (Res (None, template, Right outputs))

let answer hvar henv validfn = ... the same ...
```

Interaction with Continuations: Main loop

```
let run () =
 let rec loop jobqueue =
   let (k,template,henv) = try
      let henv = url_unquote_collate (read_line ()) in
      let Res (k,template,env) = match
        try Some (List.assoc (HEnv.find "klabel" henv) jobqueue)
        with Not_found -> None  with
        | None -> push_prompt p0 (fun () -> main (); failwith "n
        | Some k -> k henv in
      let henv = match k with
      | Some (klabel,_) -> HEnv.add "klabel" klabel henv
      | None -> henv in
      (k,template,henv)
    with e -> base_error_handler (Printexc.to_string e) in
  print_endline "Content-type: text/html\n";
  send_form stdout henv template; flush_all ();
  match k with  | Some job -> loop (job::jobqueue)  | None -> lo
in loop []
```

Interaction with Continuations: Main loop

```
let run () =
 let rec loop jobqueue =
   let (k,template,henv) = try
      let henv = url_unquote_collate (read_line ()) in
      let Res (k,template,env) = match
        try Some (List.assoc (HEnv.find "klabel" henv) jobqueue)
        with Not_found -> None  with
        | None -> push_prompt p0 (fun () -> main (); failwith "n
        | Some k -> k henv in
      let henv = match k with
      | Some (klabel,_) -> HEnv.add "klabel" klabel henv
      | None -> henv in
      (k,template,henv)
    with e -> base_error_handler (Printexc.to_string e) in
  print_endline "Content-type: text/html\n";
  send_form stdout henv template; flush_all ();
  match k with  | Some job -> loop (job::jobqueue)  | None -> lo
in loop []
```

Interaction with Continuations: Main loop

```
let run () =
 let rec loop jobqueue =
   let (k,template,henv) = try
      let henv = url_unquote_collate (read_line ()) in
      let Res (k,template,env) = match
        try Some (List.assoc (HEnv.find "klabel" henv) jobqueue)
        with Not_found -> None  with
        | None -> push_prompt p0 (fun () -> main (); failwith "n
        | Some k -> k henv in
      let henv = match k with
      | Some (klabel,_) -> HEnv.add "klabel" klabel henv
      | None -> henv in
      (k,template,henv)
    with e -> base_error_handler (Printexc.to_string e) in
  print_endline "Content-type: text/html\n";
  send_form stdout henv template; flush_all ();
  match k with  | Some job -> loop (job::jobqueue)  | None -> lo
in loop []
```

Interaction with Continuations: Main loop

```
let run () =
 let rec loop jobqueue =
   let (k,template,henv) = try
      let henv = url_unquote_collate (read_line ()) in
      let Res (k,template,env) = match
        try Some (List.assoc (HEnv.find "klabel" henv) jobqueue)
        with Not_found -> None  with
        | None -> push_prompt p0 (fun () -> main (); failwith "n
        | Some k -> k henv in
      let henv = match k with
      | Some (klabel,_) -> HEnv.add "klabel" klabel henv
      | None -> henv in
      (k,template,henv)
    with e -> base_error_handler (Printexc.to_string e) in
  print_endline "Content-type: text/html\n";
  send_form stdout henv template; flush_all ();
  match k with  | Some job -> loop (job::jobqueue)  | None -> lo
in loop []
```

Interaction with Continuations: Main loop

```
let run () =
 let rec loop jobqueue =
   let (k,template,henv) = try
      let henv = url_unquote_collate (read_line ()) in
      let Res (k,template,env) = match
        try Some (List.assoc (HEnv.find "klabel" henv) jobqueue)
        with Not_found -> None  with
        | None -> push_prompt p0 (fun () -> main (); failwith "n
        | Some k -> k henv in
      let henv = match k with
      | Some (klabel,_) -> HEnv.add "klabel" klabel henv
      | None -> henv in
      (k,template,henv)
    with e -> base_error_handler (Printexc.to_string e) in
  print_endline "Content-type: text/html\n";
  send_form stdout henv template; flush_all ();
  match k with  | Some job -> loop (job::jobqueue)  | None -> lo
in loop []
```

Interaction with Persistent Continuations

```
type cgi_result' =
   Res of string * (henv,(hvar * string) list) either
type cgi_result = unit -> cgi_result'
type k_t = henv -> cgi_result

let do_inquire template env =
 let sendk (k : k_t) () =
  let fname  = Filename.temp_file "kcgi" "" in
  let klabel = Filename.basename fname in
  let () = save_state_file fname (Obj.repr k) in
  Res (template,add "klabel" klabel env) in
 let henv = shift p0 sendk in
 HEnv.add hvar_template_name template henv
```

Interaction with Persistent Continuations

```
type cgi_result' =
   Res of string * (henv,(hvar * string) list) either
type cgi_result = unit -> cgi_result'
type k_t = henv -> cgi_result

let do_inquire template env =
 let sendk (k : k_t) () =
  let fname  = Filename.temp_file "kcgi" "" in
  let klabel = Filename.basename fname in
  let () = save_state_file fname (Obj.repr k) in
  Res (template,add "klabel" klabel env) in
 let henv = shift p0 sendk in
 HEnv.add hvar_template_name template henv
```

Interaction with Persistent Continuations

```
type cgi_result' =
   Res of string * (henv,(hvar * string) list) either
type cgi_result = unit -> cgi_result'
type k_t = henv -> cgi_result

let do_inquire template env =
 let sendk (k : k_t) () =
  let fname  = Filename.temp_file "kcgi" "" in
  let klabel = Filename.basename fname in
  let () = save_state_file fname (Obj.repr k) in
  Res (template,add "klabel" klabel env) in
 let henv = shift p0 sendk in
 HEnv.add hvar_template_name template henv
```

Main Loop with Persistent Continuations

```
let run () =
  let (template,henv) =
    try
      let henv = get_form_env () in
      let Res (template,env) =
        match
          try Some (locate_cont (HEnv.find "klabel" henv))
            with Not_found -> None | Sys_error _ -> None with
          | None -> push_prompt our_p0 (fun () -> main (); failw
          | Some k -> k henv () in
        (template,henv)
      with e -> base_error_handler (Printexc.to_string e) in
    print_endline "Content-type: text/html\n";
    send_form stdout henv template; flush_all ();
    exit 0
```

Main Loop with Persistent Continuations

```
let run () =
  let (template,henv) =
    try
      let henv = get_form_env () in
      let Res (template,env) =
        match
          try Some (locate_cont (HEnv.find "klabel" henv))
            with Not_found -> None | Sys_error _ -> None with
          | None -> push_prompt our_p0 (fun () -> main (); failw
          | Some k -> k henv () in
        (template,henv)
      with e -> base_error_handler (Printexc.to_string e) in
    print_endline "Content-type: text/html\n";
    send_form stdout henv template; flush_all ();
    exit 0
```

Demo

The back button and the multiple windows

`ls -lt /tmp/kcgi*`, note the timestamps and the sizes of saved continuations

Running example

```
let main () =
 let henv = inquire "currency_read_rate.html" [] in
 let curr_name = answer "curr-name" henv vstring in
 let curr_rate = answer "rate" henv vfloat in
 let henv = inquire "currency_read_yen.html"
                    [("curr-name",curr_name)] in
 let amount = answer "curr-amount" henv vfloat in
 let yen_amount = amount /. curr_rate in
   inquire_finish "currency_result.html"
   [("curr-name",curr_name);("rate",string_of_float curr_rate);
    ("curr-amount",string_of_float amount);
    ("yen-amount", string_of_float yen_amount)];
   exit 0                                  (* unreachable *)
```

Advantages of delimited continuations

Persistent continuations are twice delimited – in control and data

- makes them small
- makes them *possible*
- makes them correct

The thread-local scope ('thread+offspring') arises naturally and requires no implementation

Ease of use

- Unmodified OCaml
- Unmodified web server (e.g., Apache)
- vs. custom Java-based CPS Scheme interpreter and web server

Outline

Design of a simple blog

```
let main () =
  let henv = inquire "blog_login.html" [] in
  let username = answer "username" henv vstring in
  let ()       = answer "password" henv ...
  let rec loop_browse () =
    let content = read_blog () in
    let henv = inquire "blog_view.html" (("blog-data",content)::
    if answer "logout" henv vbool then inquire_finish "blog_logo
    else
      if not (answer "new" henv vbool) then loop_browse () else
      let henv = inquire "blog_new.html" env in
      let rec loop_edit henv =
        if answer "cancel" henv vbool then loop_browse () else
        let title = answer "title" henv vstring in
        let body  = answer "body" henv vstring in
        let new_post = markup username title body in
        if answer "submit" henv vbool then
          let () = write_blog new_post in loop_browse ()
        else let henv = inquire "blog_new.html" ([("title",title
          loop_edit henv
    in loop_edit henv
```

Design of a simple blog

```
let main () =
  let henv = inquire "blog_login.html" [] in
  let username = answer "username" henv vstring in
  let ()       = answer "password" henv ...
  let rec loop_browse () =
    let content = read_blog () in
    let henv = inquire "blog_view.html" (("blog-data",content)::
    if answer "logout" henv vbool then inquire_finish "blog_logo
    else
      if not (answer "new" henv vbool) then loop_browse () else
      let henv = inquire "blog_new.html" env in
      let rec loop_edit henv =
        if answer "cancel" henv vbool then loop_browse () else
        let title = answer "title" henv vstring in
        let body  = answer "body" henv vstring in
        let new_post = markup username title body in
        if answer "submit" henv vbool then
          let () = write_blog new_post in loop_browse ()
        else let henv = inquire "blog_new.html" ([("title",title
          loop_edit henv
      in loop_edit henv
```

Demo of the blog

1. Login
2. Enter a new article (subject ‘Takao-san’, text ‘great hike’), submit
3. Enter another article (subject ‘Summit’, text ‘many people’)
4. Preview, go back, edit (place ’!’ in the body), preview, optionally go back, edit again, finally submit
5. Go back to one of the previous pages of editing and previewing the second article. An attempt to press any of the buttons brings the main screen. The second article, once submitted, cannot be resubmitted
6. Duplicate the window (tab)
7. In one window, enter a new article (subject ‘Way back’, text ‘slow’), preview, don’t submit
8. In the other tab, enter a new article (subject ‘Nature course’, text ‘narrow, dark, wonderful’), preview, submit, logout
9. Go back to the first tab still previewing another article. An attempt to submit brings back the login screen: the closed outer transaction invalidates all inner ones

Simple blog as a transactional CGI script

```
let rec main () =
  let henv = inquire "blog_login.html" [] in
  let username = answer "username" henv vstring in
  let ()       = answer "password" henv ...
  let env = [("username",username)] in
  let edit env = ... in
  try
  in_transaction env (fun env ->        (* user session tx *)
    let rec loop_browse () =
      let content = read_blog () in
      let henv = inquire "blog_view.html" (("blog-data",content)
      if answer "logout" henv vbool then ()
      else if not (answer "new" henv vbool) then loop_browse ()
      match (try in_transaction env edit with TX_Gone -> None) w
      | None -> loop_browse ()
      | Some new_post -> write_blog new_post; loop_browse ()
    in loop_browse ());
  inquire_finish "blog_logout.html" env
  with TX_Gone -> main ()
```

Simple blog as a transactional CGI script

```
let rec main () =
  let henv = inquire "blog_login.html" [] in
  let username = answer "username" henv vstring in
  let ()       = answer "password" henv ...
  let env = [("username",username)] in
  let edit env = ... in
  try
  in_transaction env (fun env ->        (* user session tx *)
    let rec loop_browse () =
      let content = read_blog () in
      let henv = inquire "blog_view.html" (("blog-data",content)
      if answer "logout" henv vbool then ()
      else if not (answer "new" henv vbool) then loop_browse ()
      match (try in_transaction env edit with TX_Gone -> None) w
      | None -> loop_browse ()
      | Some new_post -> write_blog new_post; loop_browse ()
    in loop_browse ());
  inquire_finish "blog_logout.html" env
  with TX_Gone -> main ()
```

The new post editing function

```
let edit env =
  let henv = inquire "blog_new.html" env in
  if answer "cancel" henv vbool then None else (* rollback *)
  let title = answer "title" henv vstring in
  let body  = answer "body" henv vstring in
  let new_post = markup username title body in
  if answer "submit" henv vbool then Some new_post (* commit *)
  else
    let () = assert (answer "preview" henv vbool) in
    let henv = inquire "blog_preview.html" (("new-post",new_post
    if answer "submit" henv vbool then Some new_post (* commit *
    else None
```

Conclusions

First implementation of persistent twice-delimited continuations in OCaml

Persistent delimited continuations are the natural fit for CGI programming

CGI script $\equiv$ console application

with differently implemented IO primitives

- natural dialogue
- lexical scoping, exception handling
- mutable data, if necessary

Delimited continuations are concrete and clickable