Vrije Universiteit Brussel

Software Languages.Lab

Programming Language Engineering Master of Computer Science

Faculty of Science and Bio-Engineering Sciences
Vrije Universiteit Brussel

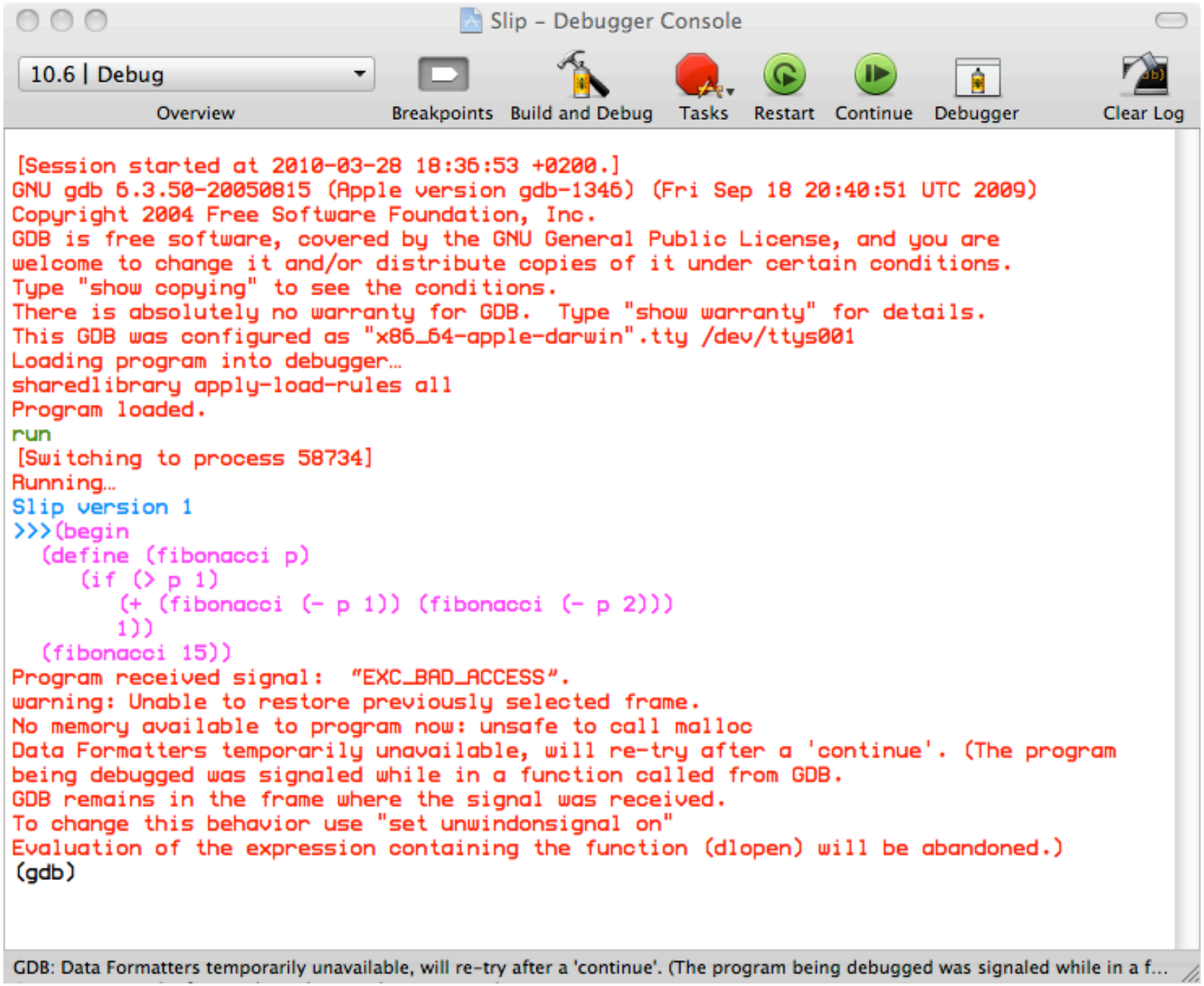
Section 6: Trampolines

Theo D'Hondt
Software Languages Lab

"… a device known as a trampoline, a piece of code that repeatedly calls functions…"

C runtime stack overflow

version 1



```
[Session started at 2010-03-28 18:36:53 +0200.]
GNU gdb 6.3.50-20050815 (Apple version gdb-1346) (Fri Sep 18 20:40:51 UTC 2009)
Copyright 2004 Free Software Foundation, Inc.
GDB is free software, covered by the GNU General Public License, and you are
welcome to change it and/or distribute copies of it under certain conditions.
Type "show copying" to see the conditions.
There is absolutely no warranty for GDB.  Type "show warranty" for details.
This GDB was configured as "x86_64-apple-darwin".tty /dev/ttys001
Loading program into debugger…
sharedlibrary apply-load-rules all
Program loaded.
run
[Switching to process 58734]
Running…
Slip version 1
>>>(begin
  (define (fibonacci p)
     (if (> p 1)
        (+ (fibonacci (- p 1)) (fibonacci (- p 2)))
        1))
  (fibonacci 15))
Program received signal:  “EXC_BAD_ACCESS”.
warning: Unable to restore previously selected frame.
No memory available to program now: unsafe to call malloc
Data Formatters temporarily unavailable, will re-try after a 'continue'. (The program
being debugged was signaled while in a function called from GDB.
GDB remains in the frame where the signal was received.
To change this behavior use "set unwindonsignal on"
Evaluation of the expression containing the function (dlopen) will be abandoned.)
(gdb)
```

GDB: Data Formatters temporarily unavailable, will re-try after a 'continue'. (The program being debugged was signaled while in a f...

Slip REP-loop

setjmp.h

```
typedef enum { Initiate_trampoline  = 0,
               Continue_trampoline  = 1,
               Terminate_trampoline = 2 } TRA_type;

typedef jmp_buf EXI_type;

static EXI_type Trampoline;
```

```
    Continue_print_function = make_CFN(continue_print);
    Print_continuation = make_CNT(Continue_print_function,
                                  Main_Empty_Continuation,
                                  0);
    Slip_Print("cpSlip/c version 2");
    for(;;)
      read_eval_print();
```

```
static NIL_type
                                CNT_type Continuation,
                                ENV_type Environment)
  { Print_Print(Value);
    Dictionary_Environment = Environment;
    longjmp(Trampoline,
            Terminate_trampoline); }
```

version 2

Slip REP-loop

```
typedef enum { Initiate_trampoline  = 0,
               Continue_trampoline  = 1,
               Terminate_trampoline = 2 } TRA_type;

typedef jmp_buf EXI_type;

static EXI_type Trampoline;
```

setjmp.h

```
    Continue_print_function = make_CFN(continue_print);
    Print_continuation = make_CNT(Continue_print_function,
                                  Main_Empty_Continuation,
                                  0);
    Slip_Print("cpSlip/c version 2");
    for(;;)
      read_eval_print();
```

```
static NIL_type
                                    CNT_type Continuation,
                                    ENV_type Environment)
  { Print_Print(Value);
    Dictionary_Environment = Environment;
    longjmp(Trampoline,
            Terminate_trampoline); }
```

version 2

Slip REP-loop

```
typedef enum { Initiate_trampoline  = 0,
               Continue_trampoline  = 1,
               Terminate_trampoline = 2 } TRA_type;

typedef jmp_buf EXI_type;

static EXI_type Trampoline;
```

setjmp.h

```
    Continue_print_function = make_CFN(continue_print);
    Print_continuation = make_CNT(Continue_print_function,
                                  Main_Empty_Continuation,
                                  0);
    Slip_Print("cpSlip/c version 2");
    for(;;)
      read_eval_print();
```

```
static NIL_type
                                CNT_type Continuation,
                                ENV_type Environment)
  { Print_Print(Value);
    Dictionary_Environment = Environment;
    longjmp(Trampoline,
            Terminate_trampoline); }
```

version 2

Slip REP-loop

```
typedef enum { Initiate_trampoline  = 0,
               Continue_trampoline  = 1,
               Terminate_trampoline = 2 } TRA_type;

typedef jmp_buf EXI_type;

static EXI_type Trampoline;
```

setjmp.h

```
    Continue_print_function = make_CFN(continue_print);
    Print_continuation = make_CNT(Continue_print_function,
                                  Main_Empty_Continuation,
                                  0);
    Slip_Print("cpSlip/c version 2");
    for(;;)
      read_eval_print();
```

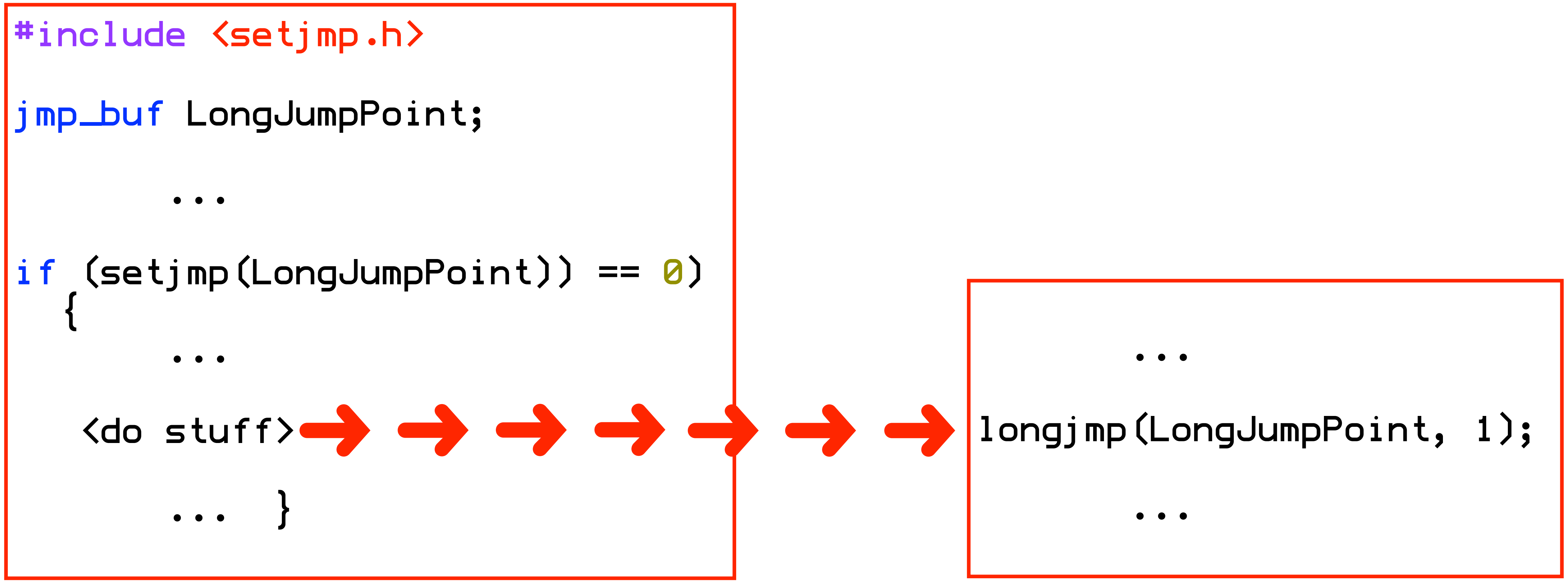
```
static NIL_type continue_print(EXP_type Value,
                               CNT_type Continuation,
                               ENV_type Environment)
  { Print_Print(Value);
    Dictionary_Environment = Environment;
    longjmp(Trampoline,
            Terminate_trampoline); }
```

version 2

C “long jumps”

```
#include <setjmp.h>

jmp_buf LongJumpPoint;

       ...

if (setjmp(LongJumpPoint)) == 0)
  {
       ...

   <do stuff>

       ...  }
```

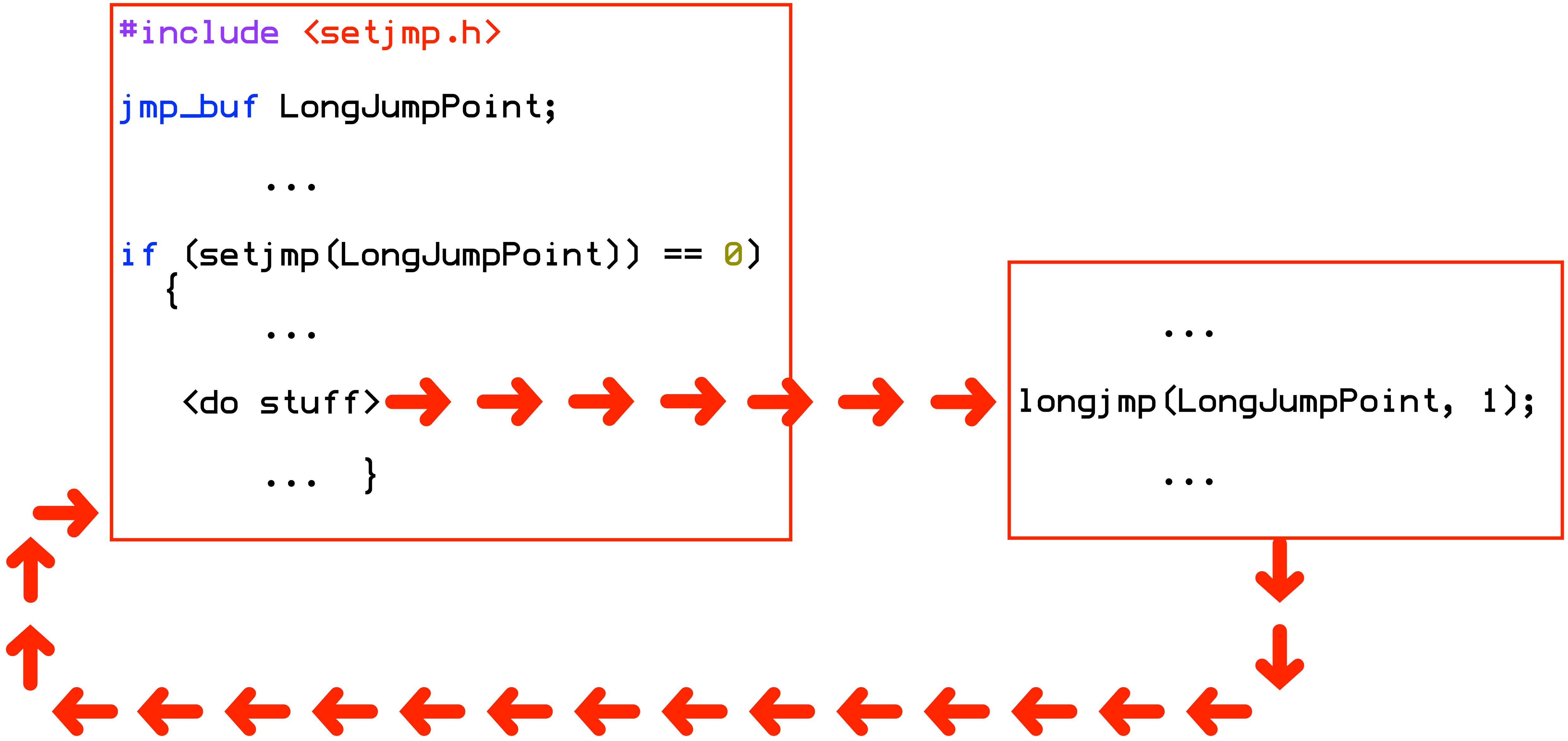
C “long jumps”



```
#include <setjmp.h>

jmp_buf LongJumpPoint;

       ...

if (setjmp(LongJumpPoint)) == 0)
  {
       ...

    <do stuff>

       ...  }
```

```
       ...

longjmp(LongJumpPoint, 1);

       ...
```

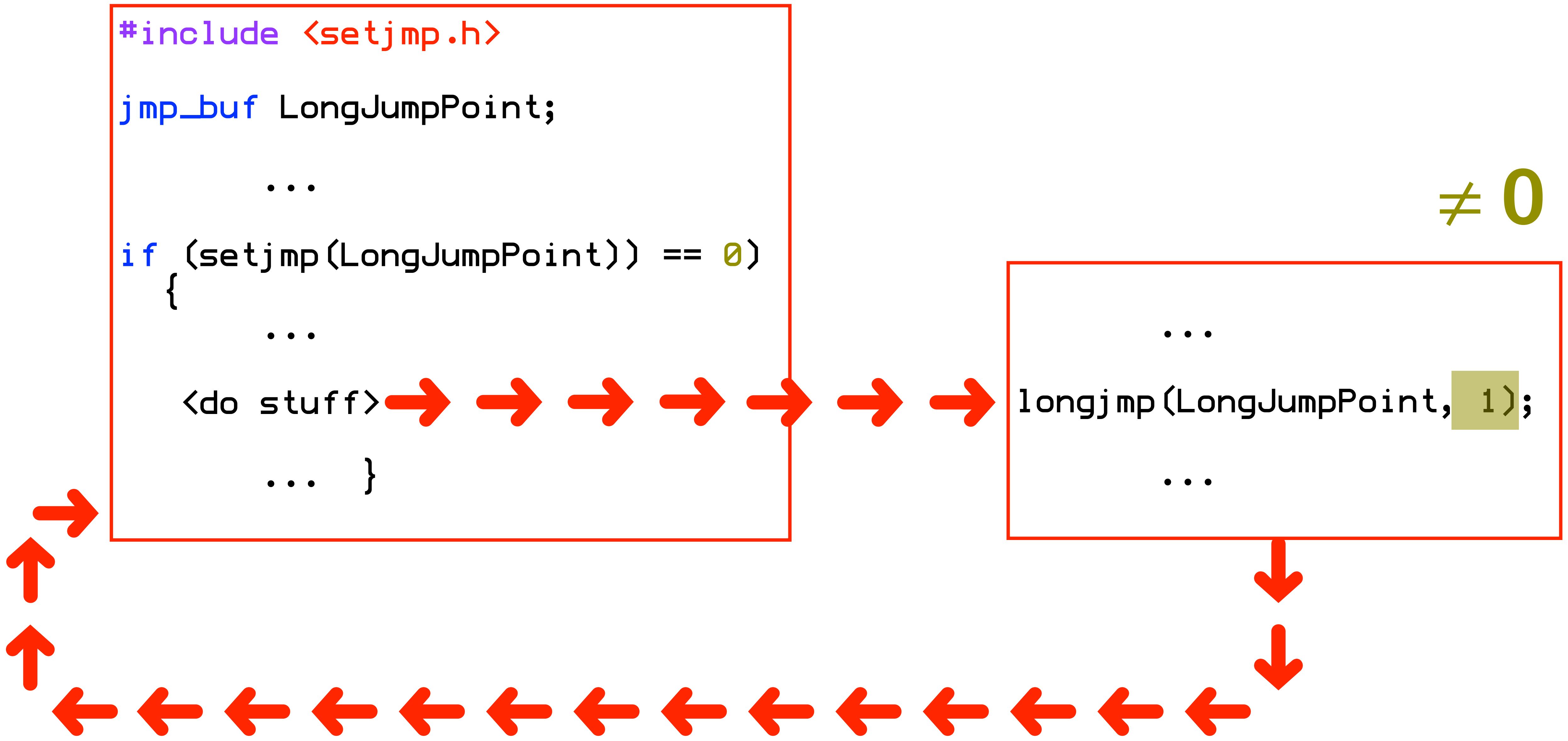
C “long jumps”



```
#include <setjmp.h>

jmp_buf LongJumpPoint;

       ...

if (setjmp(LongJumpPoint)) == 0)
  {
       ...

    <do stuff>

       ...  }
```

```
       ...

longjmp(LongJumpPoint, 1);

       ...
```

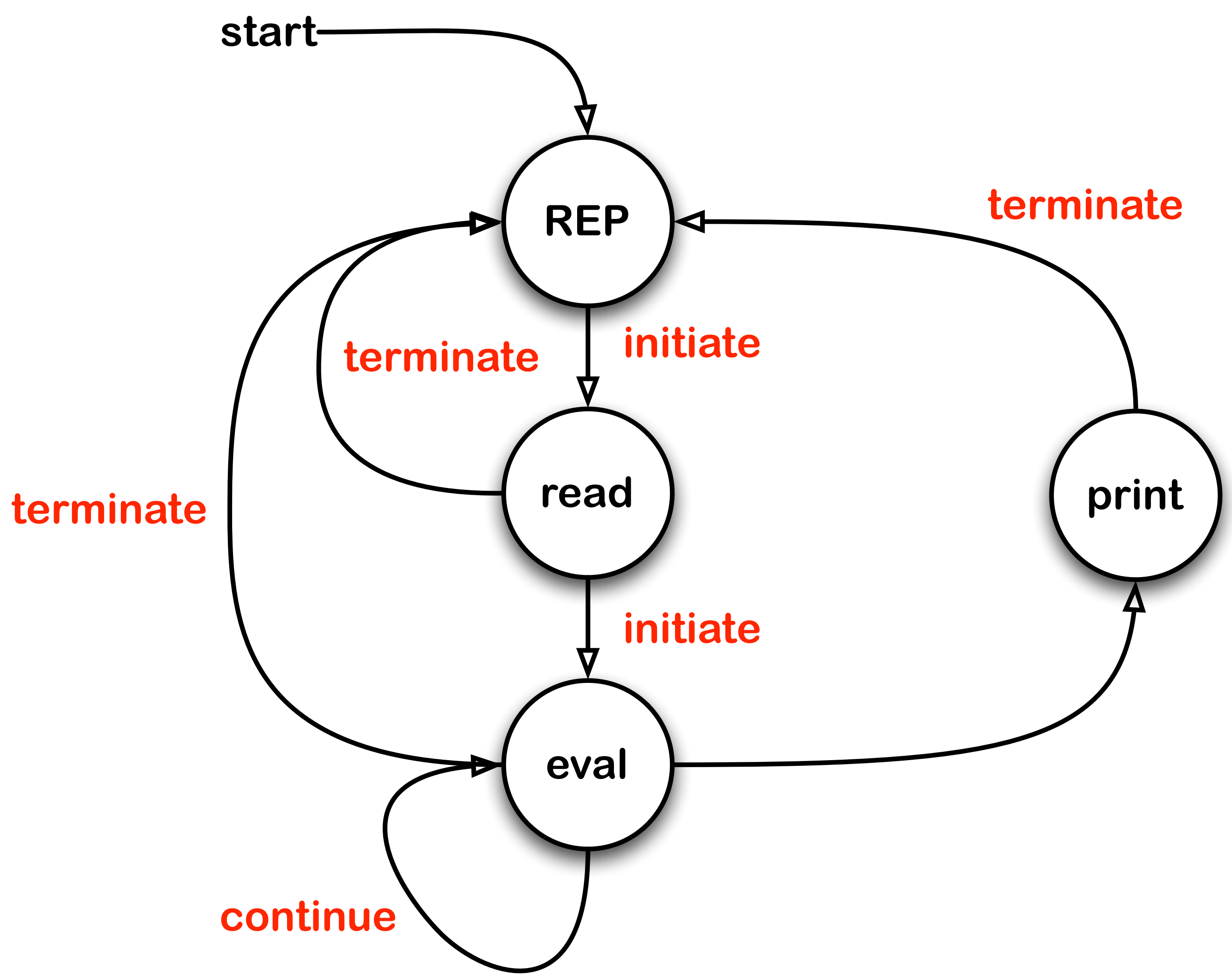
C “long jumps”



```
#include <setjmp.h>

jmp_buf LongJumpPoint;

      ...

if (setjmp(LongJumpPoint)) == 0)
  {
      ...
    <do stuff>
      ...  }
```

```
      ...
longjmp(LongJumpPoint, 1);
      ...
```

The Trampoline automaton



The Trampoline automaton (cont'd)

```
{ Initiate_trampoline  = 0,
  Continue_trampoline  = 1,
  Terminate_trampoline = 2 }
```

```
EXP_type Main_Error(TXT_type Error)
  { Slip_Print(Error);
    longjmp(Trampoline, Terminate_trampoline);
    return Main_Unspecified; }
```

```
static NIL_type read_eval_print(NIL_type)
  { TRA_type status;
    TXT_type input;
    Slip_Print("\n>>>");
    Slip_Read(&input);
    if ((status = setjmp(Trampoline)) == Initiate_trampoline)
      { Trampoline_expression   = Read_Parse(input);
        Trampoline_continuation = Print_continuation;
        Trampoline_environment  = Dictionary_Environment;
        do
          { if ((status = setjmp(Trampoline)) == Initiate_trampoline)
              Evaluate_Evaluate(Trampoline_expression,
                                Trampoline_continuation,
                                Trampoline_environment); }
        while (status == Continue_trampoline); }}
```

The Trampoline automaton (cont'd)

```
{ Initiate_trampoline  = 0,
  Continue_trampoline  = 1,
  Terminate_trampoline = 2 }
```

```
EXP_type
  { Slip_
    longj
    retur
```

```
NIL_type Main_Exit(EXP_type Expression,
                   CNT_type Continuation,
                   ENV_type Environment)
  { Trampoline_expression   = Expression;
    Trampoline_continuation = Continuation;
    Trampoline_environment  = Environment;
    longjmp(Trampoline, Continue_trampoline); }
```

```
static NIL_type read_eval_print(NIL_type)
  { TRA_type status;
    TXT_type input;
    Slip_Print("\n>>>");
    Slip_Read(&input);
    if ((status = setjmp(Trampoline)) == Initiate_trampoline)
      { Trampoline_expression   = Read_Parse(input);
        Trampoline_continuation = Print_continuation;
        Trampoline_environment  = Dictionary_Environment;
        do
          { if ((status = setjmp(Trampoline)) == Initiate_trampoline)
              Evaluate_Evaluate(Trampoline_expression,
                                Trampoline_continuation,
                                Trampoline_environment); }
        while (status == Continue_trampoline); }}
```

The Trampoline automaton (cont'd)

```
{ Initiate_trampoline  = 0,
  Continue_trampoline  = 1,
  Terminate_trampoline = 2 }
```

```
EXP_type
  { Slip_
    longj
    retur
```

```
NIL_type Main_Exit(EXP_type Expression,
                   CNT_type Continuation,
                   ENV_type Environment)
  { Trampoline_expression   = Expression;
    Trampoline_continuation = Continuation;
    Trampoline_environment  = Environment;
    longjmp(Trampoline, Continue_trampoline); }
```

```
static NIL_type read_eval_print(NIL_type)
  { TRA_type status;
    TXT_type input;
    Slip_Print("\n>>>");
    Slip_Read(&input);
    if ((status = setjmp(Trampoline)) == Initiate_trampoline)
      { Trampoline_expression   = Read_Parse(input);
        Trampoline_continuation = Print_continuation;
        Trampoline_environment  = Dictionary_Environment;
        do
          { if ((status = setjmp(Trampoline)) == Initiate_trampoline)
              Evaluate_Evaluate(Trampoline_expression,
                                Trampoline_continuation,
                                Trampoline_environment); }
        while (status == Continue_trampoline); }}
```

globals

Invoking the Trampoline

```
static NIL_type evaluate_expression(EXP_type Expression,
                                    CNT_type Continuation,
                                    ENV_type Environment)
  { TAG_type tag;
    if (--Exit_counter == 0)
      { Exit_counter = Exit_counter_preset;
        Main_Exit(Expression,
                  Continuation,
                  Environment); }
    tag = Tag_of(Expression);
    switch (tag)
      { case PAI_tag:
          evaluate_form(Expression,
                        Continuation,
                        Environment);
```

Invoking the Trampoline

```
static NIL_type evaluate_expression(EXP_type Expression,
                                    CNT_type Continuation,
                                    ENV_type Environment)
  { TAG_type tag;
    if (--Exit_counter == 0)
      { Exit_counter = Exit_counter_preset;
        Main_Exit(Expression,
                  Continuation,
                  Environment); }
    tag = Tag_of(Expression);
    switch (tag)
      { case PAI_tag:
            evaluate_form(Expression,
                          Continuation,
                          Environment);
```

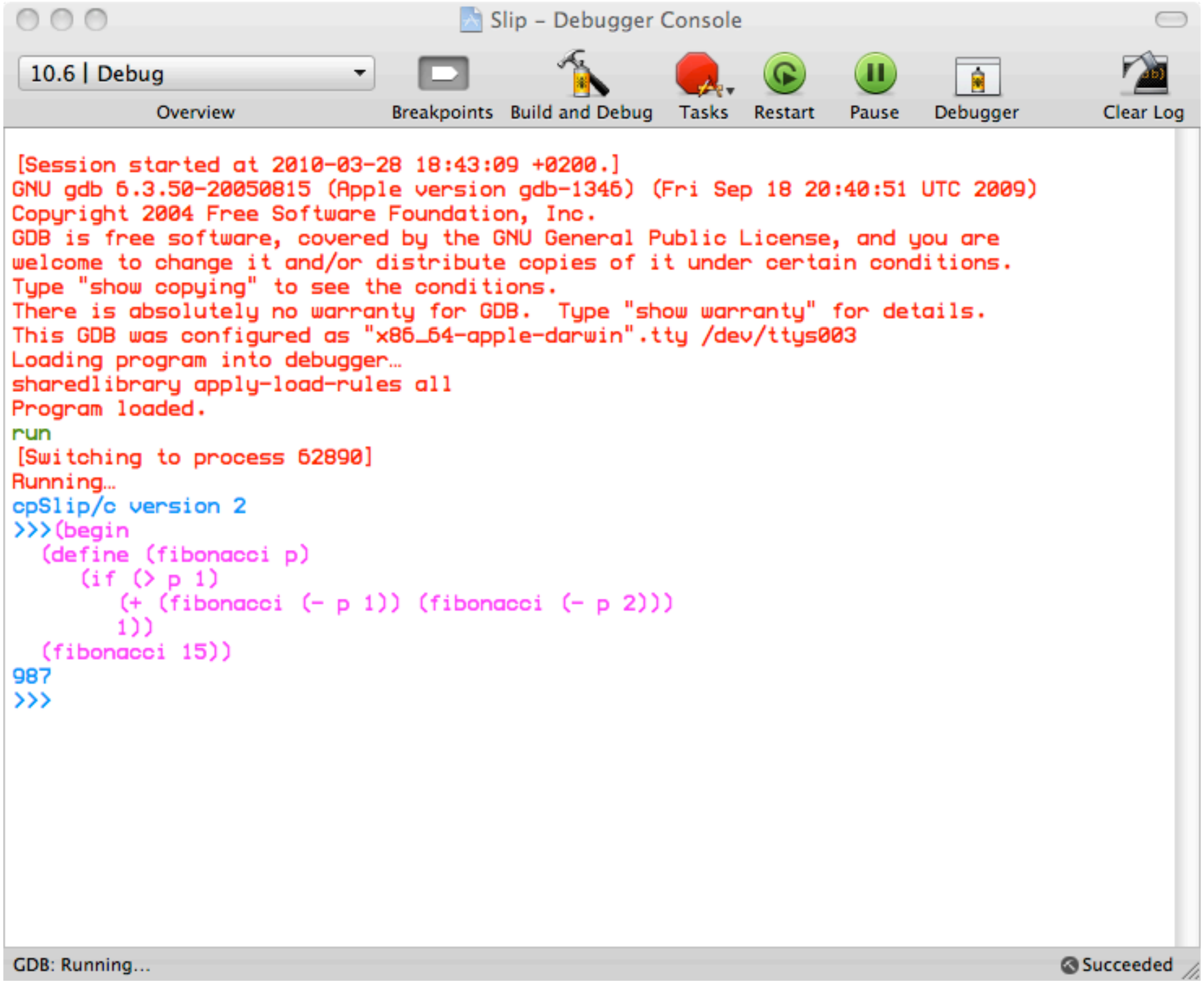
exit from loop

Invoking the Trampoline

```
static NIL_type evaluate_expression(EXP_type Expression,
                                    CNT_type Continuation,
                                    ENV_type Environment)
  { TAG_type tag;
    if (--Exit_counter == 0)
      { Exit_counter = Exit_counter_preset;
        Main_Exit(Expression,
                  Continuation,
                  Envir
    tag = Tag_of(Expres
    switch (tag)
      { case PAI_tag:
          evaluate_fo
```

```
     ...
const static UNS_type Exit_counter_preset = 256;

     ...
static UNS_type Exit_counter;

     ...
Exit_counter = Exit_counter_preset;

     ...
```

No C runtime stack overflow

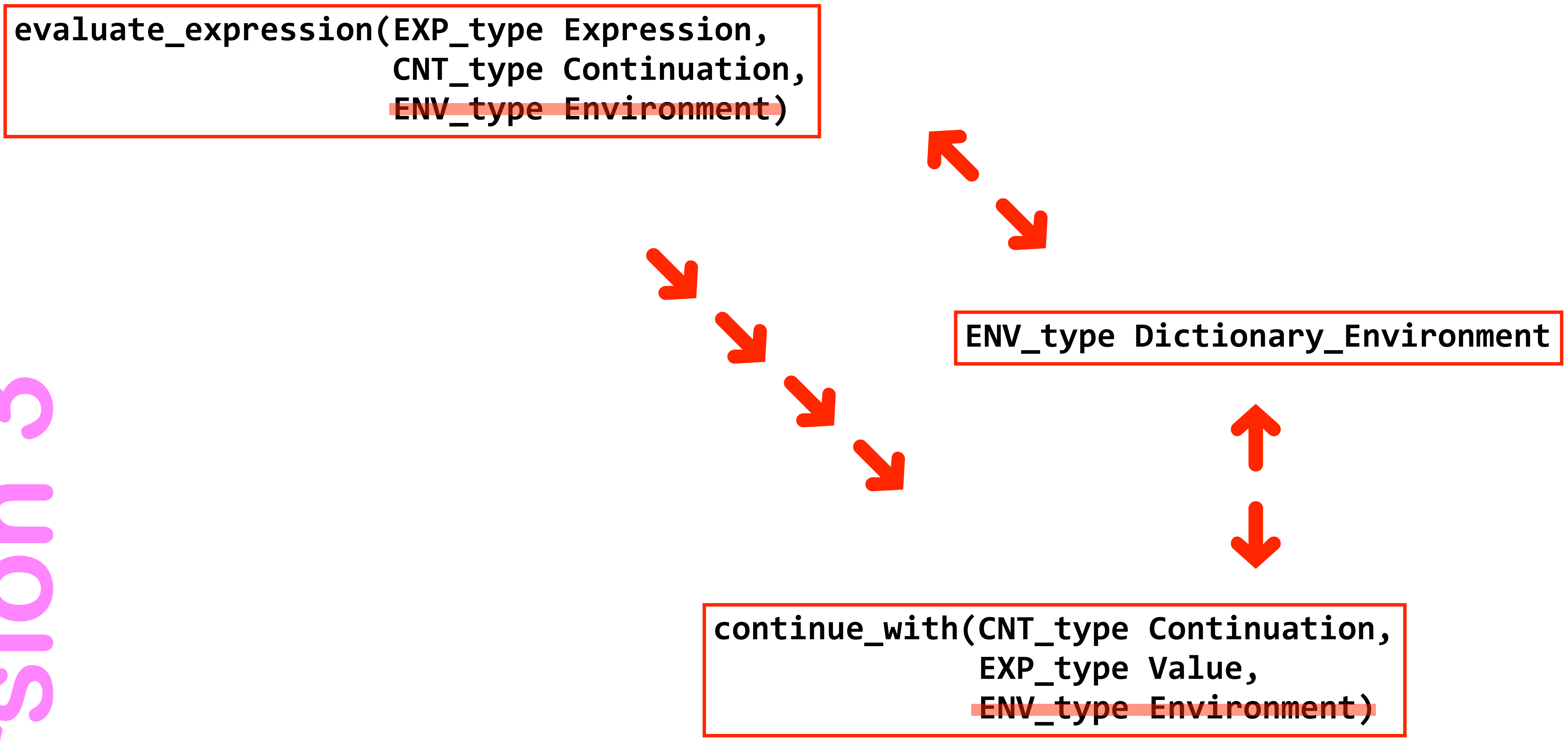


```
[Session started at 2010-03-28 18:43:09 +0200.]
GNU gdb 6.3.50-20050815 (Apple version gdb-1346) (Fri Sep 18 20:40:51 UTC 2009)
Copyright 2004 Free Software Foundation, Inc.
GDB is free software, covered by the GNU General Public License, and you are
welcome to change it and/or distribute copies of it under certain conditions.
Type "show copying" to see the conditions.
There is absolutely no warranty for GDB.  Type "show warranty" for details.
This GDB was configured as "x86_64-apple-darwin".tty /dev/ttys003
Loading program into debugger…
sharedlibrary apply-load-rules all
Program loaded.
run
[Switching to process 62890]
Running…
cpSlip/c version 2
>>>(begin
  (define (fibonacci p)
     (if (> p 1)
        (+ (fibonacci (- p 1)) (fibonacci (- p 2)))
        1))
  (fibonacci 15))
987
>>>
```

GDB: Running...

Succeeded

Factoring out the Environment



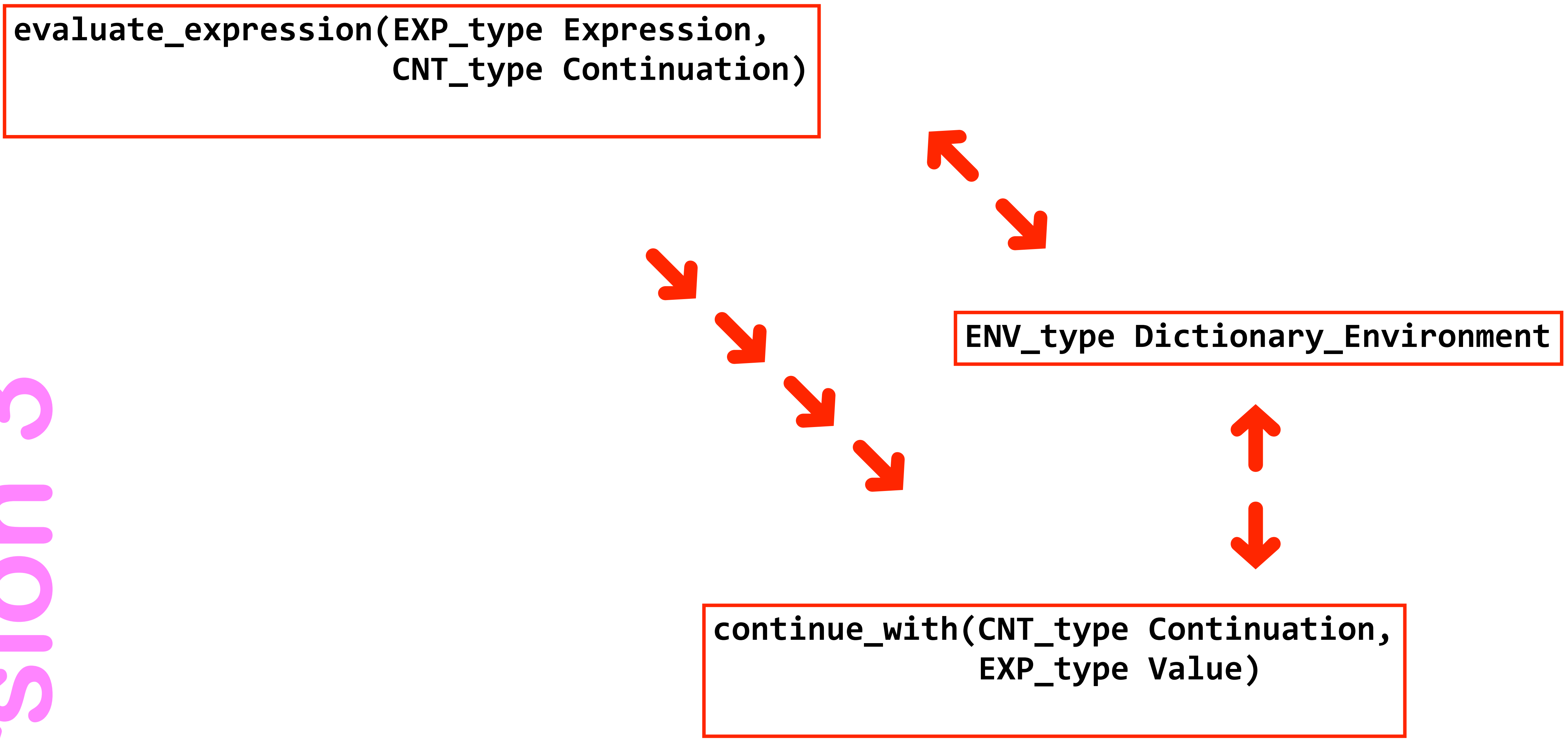
```
evaluate_expression(EXP_type Expression,
                    CNT_type Continuation,
                    ~~ENV_type Environment~~)
```

```
ENV_type Dictionary_Environment
```

```
continue_with(CNT_type Continuation,
              EXP_type Value,
              ~~ENV_type Environment~~)
```

version 3

Factoring out the Environment



```
evaluate_expression(EXP_type Expression,
                    CNT_type Continuation)
```

```
ENV_type Dictionary_Environment
```

```
continue_with(CNT_type Continuation,
              EXP_type Value)
```

version 3

Evaluating a Slip set!

```
static NIL_type evaluate_set(PAI_type Operands,
                             CNT_type Continuation)

                    if (is_NUL(residue))
                      { continuation = make_CNT(Continue_set,
                                                Continuation,
                                                sET_size);
                        set_thread = (sET_type)continuation;
                        set_thread->var = variable;
                        evaluate_expression(expression,
                                            continuation); }
```

Evaluating a Slip set! (cont'd)

```
static NIL_type continue_set(EXP_type Value,
                             CNT_type Continuation,
                             ~~ENV_type Environment)~~
  { sET_type set_thread;
    CNT_type continuation;
    SYM_type variable;
    set_thread = (sET_type)Continuation;
    continuation = set_thread->cnt;
    variable     = set_thread->var;
    Dictionary_Replace(variable,
                       Value,
                       ~~Environment~~);
    continue_with(continuation,
                  Value,
                  ~~Environment)~~; }
```

Evaluating a Slip set! (cont’d)

```
static NIL_type continue_set(EXP_type Value,
                             CNT_type Continuation)

  { sET_type set_thread;
    CNT_type continuation;
    SYM_type variable;
    set_thread = (sET_type)Continuation;
    continuation = set_thread->cnt;
    variable     = set_thread->var;
    Dictionary_Replace(variable,
                       Value);

    continue_with(continuation,
                  Value); }
```

Global Environments in function calls

```
static NIL_type evaluate_body(PRC_type Procedure,
                              CNT_type Continuation,
                              ENV_type Closure)
  { bOD_type body_thread;
    CNT_type continuation;
    PAI_type body;
    continuation = make_CNT(Continue_body,
                            Continuation,
                            bOD_size);
    body_thread = (bOD_type)continuation;
    body_thread->env = Dictionary_Environment;
    Dictionary_Environment = Closure;
    body = Procedure->bod;
    evaluate_sequence(body,
                      continuation); }

static NIL_type initialize_body(NIL_type)
  { Continue_body = make_CFN(continue_body); }
```

```
static CFN_type Continue_body;

typedef struct bOD * bOD_type;
typedef struct bOD { CEL_type hdr;
                     CFN_type cfn;
                     CNT_type cnt;
                     ENV_type env;

static const UNS_type bOD_size = chunk_size(bOD);

static NIL_type continue_body(EXP_type Value,
                              CNT_type Continuation)
  { bOD_type body_thread;
    CNT_type continuation;
    body_thread = (bOD_type)Continuation;
    continuation = body_thread->cnt;
    Dictionary_Environment = body_thread->env;
    continue_with(continuation,
                  Value); }
```

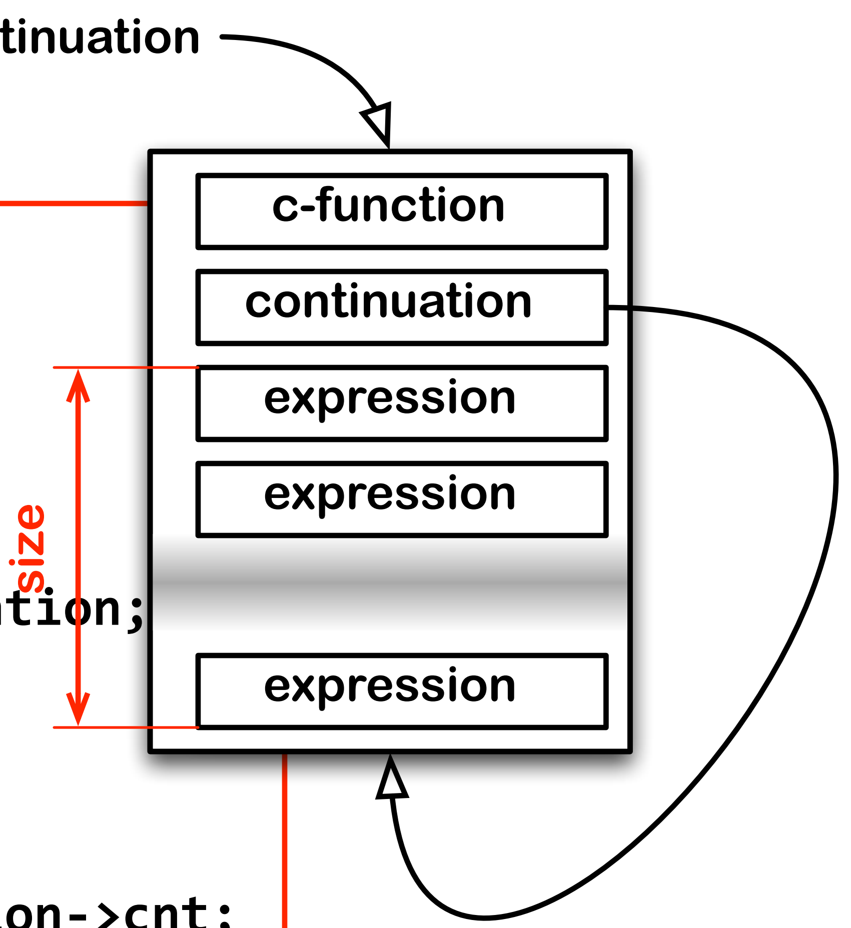
Global Environments in function calls

```
static NIL_type evaluate_body(PRC_type Procedure,
                              CNT_type Continuation,
                              ENV_type Closure)
  { bOD_type body_thread;
    CNT_type continuation;
    PAI_type body;
    continuation = make_CNT(Continue_body,
                            Continuation,
                            bOD_size);
    body_thread = (bOD_type)continuation;
    body_thread->env = Dictionary_Environment;
    Dictionary_Environment = Closure;
    body = Procedure->bod;
    evaluate_sequence(body,
                      continuation); }

static NIL_type initialize_body(NIL_type)
  { Continue_body = make_CFN(continue_body); }
```

```
static CFN_type Continue_body;

typedef struct bOD * bOD_type;
typedef struct bOD { CEL_type hdr;
                     CFN_type cfn;
                     CNT_type cnt;
                     ENV_type env;

static const UNS_type bOD_size = chunk_size(bOD);

static NIL_type continue_body(EXP_type Value,
                              CNT_type Continuation)
  { bOD_type body_thread;
    CNT_type continuation;
    body_thread = (bOD_type)Continuation;
    continuation = body_thread->cnt;
    Dictionary_Environment = body_thread->env;
    continue_with(continuation,
                  Value); }
```

environment swap in/out

Threaded continuations



```
static CNT_type Threaded_continuation;



NIL_type Thread_Initialize(NIL_type)
  { Threaded_continuation = Main_Empty_Continuation;

CNT_type Thread_Pop(NIL_type)
  { CNT_type continuation;
    continuation = Threaded_continuation;
    Threaded_continuation = Threaded_continuation->cnt;
    return continuation; }

CNT_type Thread_Push(CFN_type C_function,
                     UNS_type Size)
  { CNT_type continuation;
    continuation = make_CNT(C_function,
                            Threaded_continuation,
                            Size);
    Threaded_continuation = continuation;
    return continuation; }
```

version 4

A threaded Trampoline

```
{ Initiate_trampoline  = 0,
  Continue_trampoline  = 1,
  Terminate_trampoline = 2,
  Abort_trampoline     = 3 }
```

```
NIL_type Main_Exit(EXP_type Expression)
  { Trampoline_expression = Expression;
    longjmp(Trampoline,
            Continue_trampoline); }
```

```
static NIL_type read_eval_print(NIL_type)
  { TRA_type status;
    TXT_type input;
    Slip_Print("\n>>>");
    Slip_Read(&input);
    if ((status = setjmp(Trampoline)) == Initiate_trampoline)
      { Trampoline_expression = Read_Parse(input);
        Rollback_environment = Dictionary_Environment;
        Thread_Initialize();
        Thread_Push(Continue_print_function,
                    0);
        do
          { if ((status = setjmp(Trampoline)) == Initiate_trampoline)
              Evaluate_Evaluate(Trampoline_expression); }
        while (status == Continue_trampoline);
        if (status == Abort_trampoline)
          Dictionary_Environment = Rollback_environment; }}
```

A threaded Trampoline: set!

```
static CFN_type Continue_set;

typedef struct sET * sET_type;
typedef struct sET { CEL_type hdr;
                     CFN_type cfn;
                     CNT_type cnt;
                     SYM_type var; } sET;

static const UNS_type sET_size = chunk_size(sET);

static NIL_type continue_set(EXP_type Value,
                             CNT_type Continuation)
  { sET_type set_thread;

    continue_with(Value); }

static NIL_type evaluate_set(PAI_type Operands)
  { sET_type set_thread;

          variable = Operands->car;

            { expressions = Operands->cdr;

                      { continuation = Thread_Push(Continue_set,
                                                   sET_size);
                        set_thread = (sET_type)continuation;
                        set_thread->var = variable;
                        evaluate_expression(expression); }

                                    }

static NIL_type initialize_set(NIL_type)
  { Continue_set = make_CFN(continue_set); }
```

A threaded Trampoline: set!

```
static CFN_type Continue_set;

typedef struct sET * sET_type;
typedef struct sET { CEL_type hdr;
                     CFN_type cfn;
                     CNT_type cnt;
                     SYM_type var; } sET;

static const UNS_type sET_size = chunk_size(sET);

static NIL_type continue_set(EXP_type Value,
                             CNT_type Continuation)
  { sET_type set_thread;

    continue_with(Value); }

static NIL_type evaluate_set(PAI_type Operands)
  { sET_type set_thread;

          variable = Operands->car;

            { expressions = Operands->cdr;

                      { continuation = Thread_Push(Continue_set,
                                                   sET_size);
                        set_thread = (sET_type)continuation;
                        set_thread->var = variable;
                        evaluate_expression(expression); }

                                    }

static NIL_type initialize_set(NIL_type)
  { Continue_set = make_CFN(continue_set); }
```

```
static NIL_type continue_set(EXP_type Value,
                             CNT_type Continuation)
  { sET_type set_thread;
    SYM_type variable;
    set_thread = (sET_type)Continuation;
    variable = set_thread->var;
    Dictionary_Replace(variable,
                       Value);
    continue_with(Value); }
```

A threaded Trampoline: set!

```
static CFN_type Continue_set;

typedef struct sET * sET_type;
typedef struct sET { CEL_type hdr;
                     CFN_type cfn;
                     CNT_type cnt;
                     SYM_type var; } sET;

static const UNS_type sET_size = chunk_size(sET);

static NIL_type continue_set(EXP_type Value,
                             CNT_type Continuation)
  { sET_type set_thread;

    continue_with(Value); }

static NIL_type evaluate_set(PAI_type Operands)
  { sET_type set_thread;

          variable = Operands->car;

            { expressions = Operands->cdr;

                      { continuation = Thread_Push(Continue_set,
                                                   sET_size);
                        set_thread = (sET_type)continuation;
                        set_thread->var = variable;
                        evaluate_expression(expression); }

                               }

static NIL_type initialize_set(NIL_type)
  { Continue_set = make_CFN(continue_set); }
```

```
static NIL_type continue_set(EXP_type Value,
                             CNT_type Continuation)
  { sET_type set_thread;
    SYM_type variable;
    set_thread = (sET_type)Continuation;
    variable = set_thread->var;
    Dictionary_Replace(variable,
                       Value);
    continue_with(Value); }
```

```
static NIL_type continue_with(EXP_type Value)
  { CNT_type continuation;
    CCC_type function;
    CFN_type c_function;
    continuation = Thread_Pop();
    c_function = continuation->cfn;
    function = c_function->ccc;
    function(Value,
             continuation); }
```

A threaded Trampoline: set!

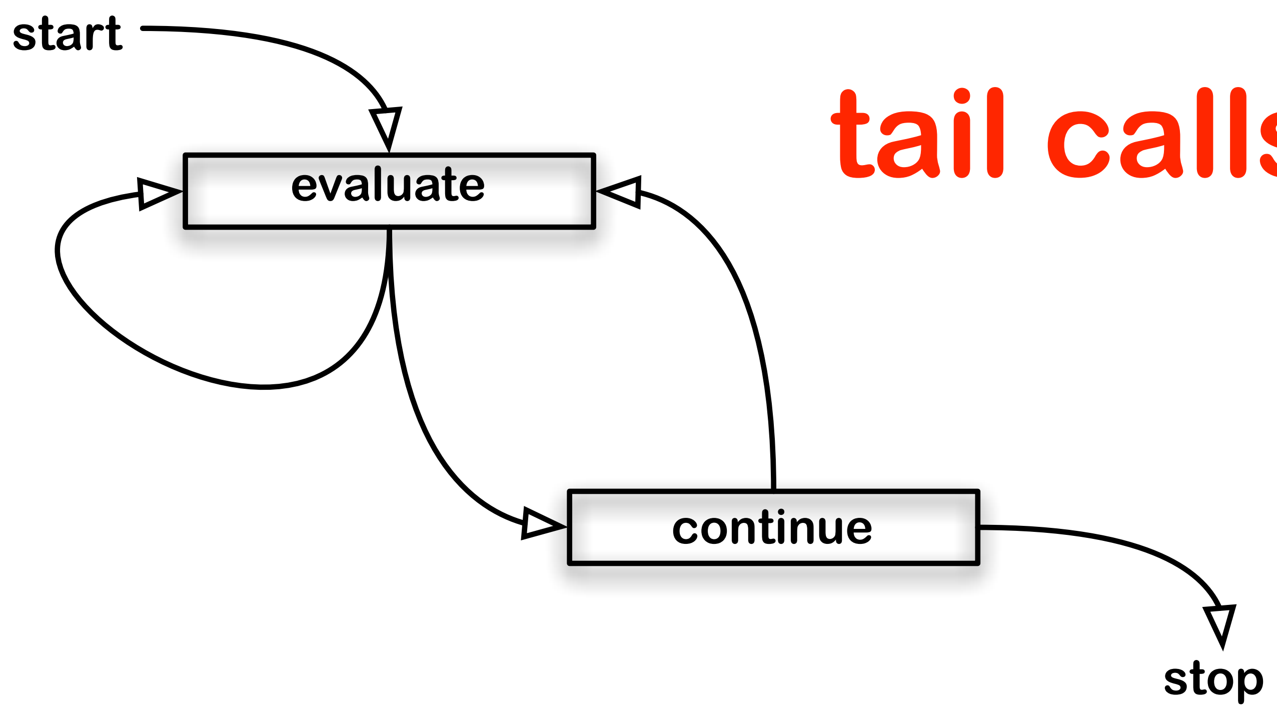
```
static CFN_type Continue_set;

typedef struct sET * sET_type;
typedef struct sET { CEL_type hdr;
                     CFN_type cfn;
                     CNT_type cnt;
                     SYM_type var; } sET;

static const UNS_type sET_size = chunk_size(sET);

static NIL_type continue_set(EXP_type Value,
                             CNT_type Continuation)
  { sET_type set_thread;

    continue_with(Value); }

static NIL_type evaluate_set(PAI_type Operands)
  { sET_type set_thread;

          variable = Operands->car;

            { expressions = Operands->cdr;

                 { continuation = Thread_Push(Continue_set,
                                              sET_size);
                   set_thread = (sET_type)continuation;
                   set_thread->var = variable;
                   evaluate_expression(expression); }

                            }

static NIL_type initialize_set(NIL_type)
  { Continue_set = make_CFN(continue_set); }
```

```
static NIL_type continue_set(EXP_type Value,
                             CNT_type Continuation)
  { sET_type set_thread;
    SYM_type variable;
    set_thread = (sET_type)Continuation;
    variable = set_thread->var;
    Dictionary_Replace(variable,
                       Value);
    continue_with(Value); }
```

```
static NIL_type continue_with(EXP_type Value)
  { CNT_type continuation;
    CCC_type function;
    CFN_type c_function;
    continuation = Thread_Pop();
    c_function = continuation->cfn;
    function = c_function->ccc;
    function(Value,
             continuation); }
```

Stack machine

Factoring out continuation calls



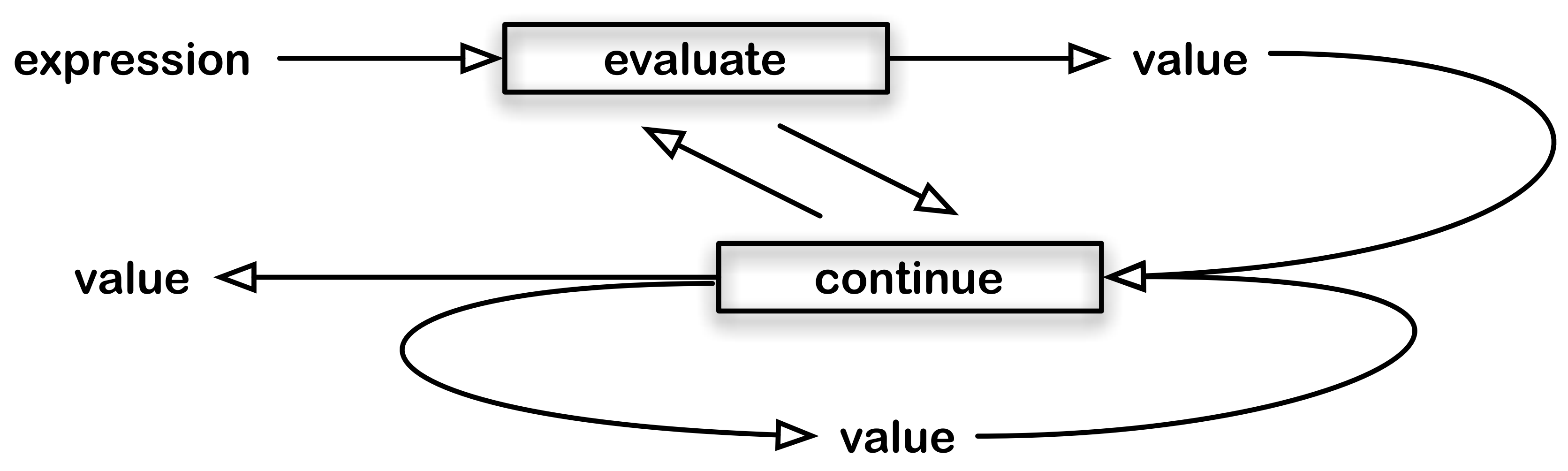
tail calls

version 4

```
do
   { if ((status = setjmp(Trampoline)) == Initiate_trampoline)
        Evaluate_Evaluate(Trampoline_expression); }
while (status == Continue_trampoline);
```

Factoring out continuation calls

function calls



version 5

```
if ((status = setjmp(Trampoline)) == Initiate_loop)
  { value = Evaluate_Evaluate(expression);
    for (;;)
      value = Evaluate_Continue(value); }
```

A functional version

```
static EXP_type evaluate_expression(EXP_type Expression)
  { TAG_type tag;
    tag = Tag_of(Expression);
    switch (tag)
      { case PAI_tag:
          return evaluate_form(Expression);
        case SYM_tag:
          return evaluate_symbol(Expression);
        case CHA_tag:
        case FLS_tag:
        case NUL_tag:
        case NBR_tag:
        case REA_tag:
        case STR_tag:
        case TRU_tag:
        case VEC_tag:
          return evaluate_value(Expression); }
    return Main_Error_Tag(IXT_error_string,
                          tag); }
```

A functional version

```
static EXP_type evaluate_expression(EXP_type Expression)
  { TAG_type tag;
    tag = Tag_of(Expression);
    switch (tag)
      { case PAI_tag:
          return evaluate_form(Expression);
        case SYM_tag:
          return evaluate_symbol(Expression);
        case CHA_tag:
        case FLS_tag:
        case NUL_tag:
        case NBR_tag:
        case REA_tag:
        case STR_tag:
        case TRU_tag:
        case VEC_tag:
          return evaluate_value(Expression); }
    return Main_Error_Tag(IXT_error_string,
                              tag); }
```

A functional version: set!

```
static CFN_type Continue_set;

typedef struct sET * sET_type;
typedef struct sET { CEL_type hdr;
                     CFN_type cfn;
                     CNT_type cnt;
                     SYM_type var; } sET;

static const UNS_type sET_size = chunk_size(sET);

static EXP_type continue_set(EXP_type Value,
                             CNT_type Continuation)
  { sET_type set_thread;

    return Value; }

static EXP_type evaluate_set(PAI_type Operands)
  { sET_type set_thread;

          variable = Operands->car;

            { expressions = Operands->cdr;

                      { continuation = Thread_Push(Continue_set,
                                                   sET_size);
                        set_thread = (sET_type)continuation;
                        set_thread->var = variable;
                        return evaluate_expression(expression); }

                                    }

static NIL_type initialize_set(NIL_type)
  { Continue_set = make_CFN(continue_set); }
```

A functional version: set!

```
static CFN_type Continue_set;

typedef struct sET * sET_type;
typedef struct sET { CEL_type hdr;
                     CFN_type cfn;
                     CNT_type cnt;
                     SYM_type var; } sET;

static const UNS_type sET_size = chunk_size(sET);

static EXP_type continue_set(EXP_type Value,
                             CNT_type Continuation)
  { sET_type set_thread;

    return Value; }

static EXP_type evaluate_set(PAI_type Operands)
  { sET_type set_thread;

          variable = Operands->car;

            { expressions = Operands->cdr;

                      { continuation = Thread_Push(Continue_set,
                                                   sET_size);
                        set_thread = (sET_type)continuation;
                        set_thread->var = variable;
                        return evaluate_expression(expression); }

                                   }

static NIL_type initialize_set(NIL_type)
  { Continue_set = make_CFN(continue_set); }
```

A functional version: set!

```
static CFN_type Continue_set;

typedef struct sET * sET_type;
typedef struct sET { CEL_type hdr;
                     CFN_type cfn;
                     CNT_type cnt;
                     SYM_type var; } sET

static const UNS_type sET_size = chunk_s

static EXP_type continue_set(EXP_type Value,
                             CNT_type Continuation)
  { sET_type set_thread;

    return Value; }

static EXP_type evaluate_set(PAI_type Operands)
  { sET_type set_thread;

          variable = Operands->car;

            { expressions = Operands->cdr;

                { continuation = Thread_Push(Continue_set,
                                             sET_size);
                  set_thread = (sET_type)continuation;
                  set_thread->var = variable;
                  return evaluate_expression(expression); }

                             }

static NIL_type initialize_set(NIL_type)
  { Continue_set = make_CFN(continue_set); }
```

```
static NIL_type continue_set(EXP_type Value,
                             CNT_type Continuation)
  { sET_type set_thread;
    SYM_type variable;
    set_thread = (sET_type)Continuation;
    variable = set_thread->var;
    Dictionary_Replace(variable,
                       Value);
    return Value; }
```

A functional version: set!

```
static CFN_type Continue_set;

typedef struct sET * sET_type;
typedef struct sET { CEL_type hdr;
                     CFN_type cfn;
                     CNT_type cnt;
                     SYM_type var; } sET

static const UNS_type sET_size = chunk_si

static EXP_type continue_set(EXP_type Value,
                             CNT_type Continuation)
  { sET_type set_thread;

    return Value; }

static EXP_type evaluate_set(PAI_type Operands)
  { sET_type set_thread;

          variable = Operands->car;

            { expressions = Operands->cdr;

                      { continuation = Thread_Push(Continue_set,
                                                   sET_size);
                        set_thread = (sET_type)continuation;
                        set_thread->var = variable;
                        return evaluate_expression(expression); }

                                   }

static NIL_type initialize_set(NIL_type)
  { Continue_set = make_CFN(continue_set); }
```

```
static NIL_type continue_set(EXP_type Value,
                             CNT_type Continuation)
  { sET_type set_thread;
    SYM_type variable;
    set_thread = (sET_type)Continuation;
    variable = set_thread->var;
    Dictionary_Replace(variable,
                       Value);
    return Value; }
```

A functional version: set!

```
static CFN_type Continue_set;

typedef struct sET * sET_type;
typedef struct sET { CEL_type hdr;
                     CFN_type cfn;
                     CNT_type cnt;
                     SYM_type var; } sET;

static const UNS_type sET_size = chunk_si

static EXP_type continue_set(EXP_type Value,
                             CNT_type Continuation)
  { sET_type set_thread;

    return Value; }

static EXP_type evaluate_set(PAI_type Operands)
  { sET_type set_thread;

          variable = Operands->car;

           { expressions = Operands->cdr;

                { continuation = Thread_Push(Continue_set,
                                             sET_size);
                  set_thread = (sET_type)continuation;
                  set_thread->var = variable;
                  return evaluate_expression(expression); }

                              }

static NIL_type initialize_set(NIL_type)
  { Continue_set = make_CFN(continue_set); }
```

```
static NIL_type continue_set(EXP_type Value,
                             CNT_type Continuation)
  { sET_type set_thread;
    SYM_type variable;
    set_thread = (sET_type)Continuation;
    variable = set_thread->var;
    Dictionary_Replace(variable,
                       Value);
    return Value; }
```

```
EXP_type Evaluate_Continue(EXP_type Value)
  { return continue_with(Value); }


static EXP_type continue_with(EXP_type Value)
  { CNT_type continuation;
    CCC_type function;
    CFN_type c_function;
    continuation = Thread_Pop();
    c_function = continuation->cfn;
    function = c_function->ccc;
    return function(Value,
                    continuation); }
```

A functional version: set!

```
static CFN_type Continue_set;

typedef struct sET * sET_type;
typedef struct sET { CEL_type hdr;
                     CFN_type cfn;
                     CNT_type cnt;
                     SYM_type var; } sET

static const UNS_type sET_size = chunk_s

static EXP_type continue_set(EXP_type Value,
                             CNT_type Continuation)
  { sET_type set_thread;

    return Value; }

static EXP_type evaluate_set(PAI_type Operands)
  { sET_type set_thread;

          variable = Operands->car;

            { expressions = Operands->cdr;

                      { continuation = Thread_Push(Continue_set,
                                                   sET_size);
                        set_thread = (sET_type)continuation;
                        set_thread->var = variable;
                        return evaluate_expression(expression); }

                                  }

static NIL_type initialize_set(NIL_type)
  { Continue_set = make_CFN(continue_set); }
```

```
static NIL_type continue_set(EXP_type Value,
                             CNT_type Continuation)
  { sET_type set_thread;
    SYM_type variable;
    set_thread = (sET_type)Continuation;
    variable = set_thread->var;
    Dictionary_Replace(variable,
                       Value);
    return Value; }
```

```
EXP_type Evaluate_Continue(EXP_type Value)
  { return continue_with(Value); }


static EXP_type continue_with(EXP_type Value)
  { CNT_type continuation;
    CCC_type function;
    CFN_type c_function;
    continuation = Thread_Pop();
    c_function = continuation->cfn;
    function = c_function->ccc;
    return function(Value,
                    continuation); }
```

A functional version: set!

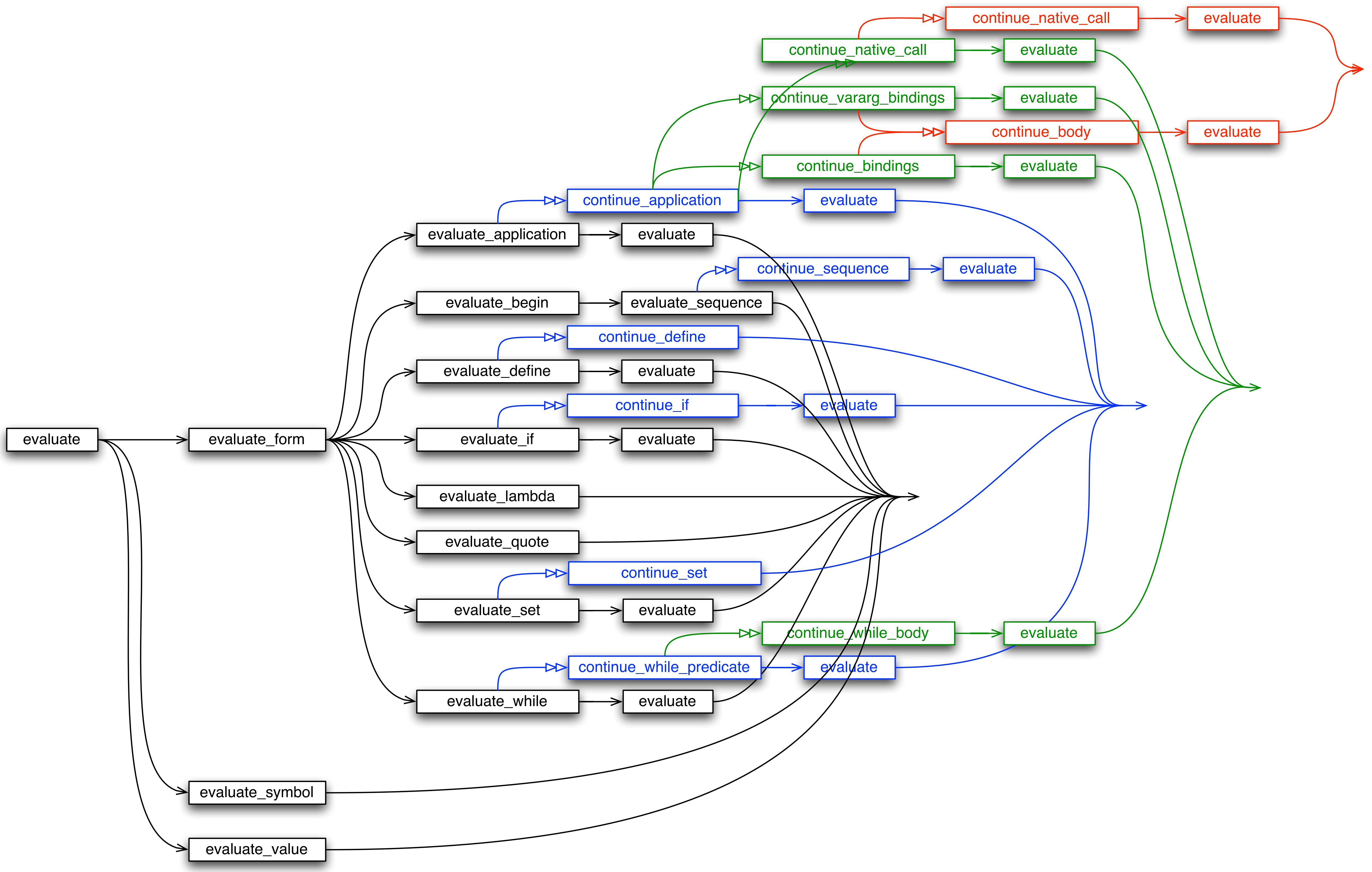
```
static CFN_type Continue_set;

typedef struct sET * sET_type;
typedef struct sET { CEL_type hdr;
                     CFN_type cfn;
                     CNT_type cnt;
                     SYM_type var; } sET;

static const UNS_type sET_size = chunk_si

static EXP_type continue_set(EXP_type Value,
                             CNT_type Continuation)
  { sET_type set_thread;

    return Value; }

static EXP_type evaluate_set(PAI_type Operands)
  { sET_type set_thread;

          variable = Operands->car;

           { expressions = Operands->cdr;

                 continuation = Thread_Push(Continue_set,
                                            sET_size);
                 set_thread = (sET_type)continuation;
                 set_thread->var = variable;
                 return evaluate_expression(expression); }

                          }

static NIL_type initialize_set(NIL_type)
  { Continue_set = make_CFN(continue_set); }
```

```
static NIL_type continue_set(EXP_type Value,
                             CNT_type Continuation)
  { sET_type set_thread;
    SYM_type variable;
    set_thread = (sET_type)Continuation;
    variable = set_thread->var;
    Dictionary_Replace(variable,
                       Value);
    return Value; }
```

```
EXP_type Evaluate_Continue(EXP_type Value)
  { return continue_with(Value); }

static EXP_type continue_with(EXP_type Value)
  { CNT_type continuation;
    CCC_type function;
    CFN_type c_function;
    continuation = Thread_Pop();
    c_function = continuation->cfn;
    function = c_function->ccc;
    return function(Value,
                    continuation); }
```

no ad-hoc trampoline

Evaluator architecture


evaluate
evaluate_form
evaluate_application
evaluate_begin
evaluate_define
evaluate_if
evaluate_lambda
evaluate_quote
evaluate_set
evaluate_while
evaluate_symbol
evaluate_value
evaluate_sequence
continue_application
continue_sequence
continue_define
continue_if
continue_set
continue_while_predicate
continue_while_body
continue_native_call
continue_vararg_bindings
continue_bindings
continue_body
evaluate