Verifying Duff's device: A simple compositional denotational semantics for Goto and computed jumps

Hendrik Tews

What is Fiasco?

- 2nd generation micro kernel, compatible with L4
- developed in the DROPS project (Dresden Real-Time Operating System)
- Verification target in the VFiasco Project (www.vfiasco.org)
- (In Robin we work on the Nova, the VFiasco successor, however, the Nova sources are not complete yet.)

The source

Thread::handle_slow_trap:

```
  ...
  if (EXPECT_FALSE (gdb_trap_recover)) goto generic_debug;
  ...
  if (!check_trap13_kernel (ts, from_user)) return 0;
  ...
  if (EXPECT_FALSE (! ((ts->cs & 3) || (ts->eflags & EFLAGS_VM)))) {
      if (check_trap13_smas (ts)) goto success;
      goto generic_debug; }
  ...
success:
  _recover_jmpbuf = 0;
  return 0;

generic_debug: ...
```

Unfortunately

- `goto` is not dead (it might be very useful)
- civilised jumps (`return`, `break`, ...) are used very often *in real code*
- occasionally rare things are used (longjmp, continuations, ...)

I. Right solution *(the mountain comes to Mohammed)*

- write all software in Haskell without jumps

II. Alternative solution *(Mohammed goes to the mountain)*

- treat jumps in semantics

Duff's device

```
void copy(int * to, int * from, int count){
  int rounds= count / 8;
  switch(count%8){
  case 0: while(rounds-- > 0){ *to++ = *from++;
  case 7:                      *to++ = *from++;
  case 6:                      *to++ = *from++;
  case 5:                      *to++ = *from++;
  case 4:                      *to++ = *from++;
  case 3:                      *to++ = *from++;
  case 2:                      *to++ = *from++;
  case 1:                      *to++ = *from++;
} }                        };
```

Something nonsensical

```
  if(a == 0) goto x;
  switch(b){
    case 0: if(c == 0)
         x:     d = 1;
              else
    case 1:     d = 2;
  }
```

Solution of Jacobs/Huismann in the LOOP project for Java

- distinguish *normal* and *abrupt* termination
- Semantic domain $S \longrightarrow Result$
- $Result$ is a disjoint union

$$Result = \begin{cases} S \uplus ok & \text{(normal termination)} \\ S \uplus break & \\ \quad \ldots & \end{cases}$$

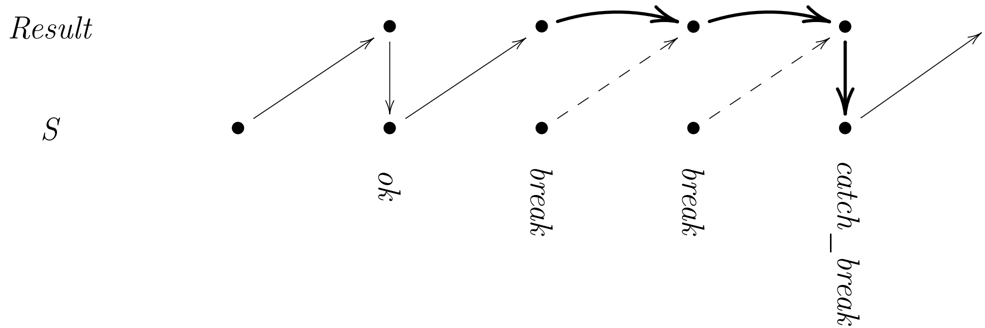
- Semantic Equations

$$[\![v := a]\!]\, s = ok(s[v := a])$$

$$[\![\texttt{break}]\!]\, s = break(s)$$

$$[\![p\ ;\ q]\!]\, s = \begin{cases} [\![q]\!]\, s' & \text{if } [\![p]\!]\, s = ok(s') \\ [\![p]\!]\, s & \text{otherwise} \end{cases}$$

$$[\![\{block\}]\!]\, s = \begin{cases} ok(s') & \text{if } [\![block]\!]\, s = ok(s') \\ ok(s') & \text{if } [\![block]\!]\, s = break(s') \\ [\![block]\!]\, s & \text{otherwise} \end{cases}$$


Result
S
ok
break
break
catch_break

structured jumps

- jump labels have a limited scope (function body)
- no jumps into other functions
- + some other restrictions
- jumps into and out of blocks/loops/if's possible

Switch

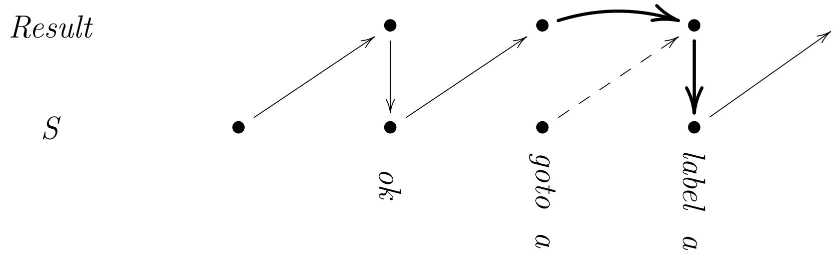
- computed jump (in contrast to a case switch)
- fall through (without `break`)
- labels may appear inside other blocks (Duff's device)

Recall nonsense:

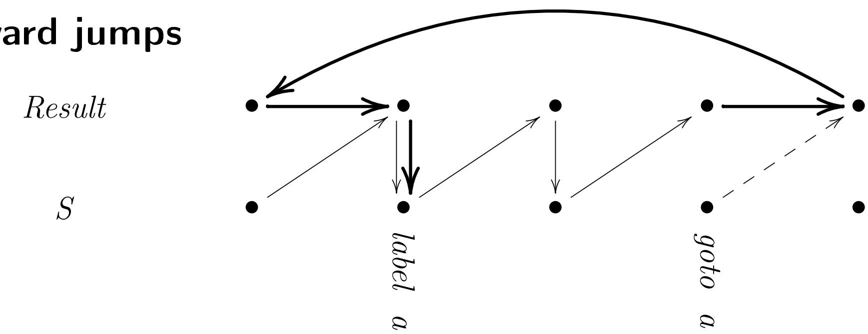
```
if(a == 0) goto x;
switch(b){
  case 0: if(c == 0)
        x:    d = 1;
             else
  case 1:     d = 2;
}
```

Extend approach of Huisman and Jacobs slightly

forward jumps



backward jumps



Semantics for Goto

- Semantic domain $Result \longrightarrow Result$
- this time

$$Result = \begin{cases} S \uplus & ok \text{ (normal termination)} \\ S \uplus & break \\ S \times \mathbb{Z} \uplus & case \\ S \uplus & default \\ S \times \mathbb{L} \uplus & goto \\ \{*\} & hang \text{ (non-termination)} \end{cases}$$

- Semantic Equations

$$[\![v = a]\!] = \left| \begin{array}{lll} ok(s) & \mapsto & ok(s[v := a]) \\ x & \mapsto & x \end{array} \right.$$

$$[\![\texttt{break}]\!] = \left| \begin{array}{lll} ok(s) & \mapsto & break(s) \\ x & \mapsto & x \end{array} \right.$$

$$[\![\texttt{goto}\ l]\!] = \left| \begin{array}{lll} ok(s) & \mapsto & goto(s, l) \\ x & \mapsto & x \end{array} \right.$$

$$[\![st_1; st_2]\!] = [\![st_2]\!] \circ [\![st_1]\!]$$

$$[\![\texttt{case}\ i :]\!] = \left|\begin{array}{lcl} case(s,i) & \mapsto & ok(s) \\ x & \mapsto & x \end{array}\right.$$

$$[\![\texttt{default} :]\!] = \left|\begin{array}{lcl} default(s) & \mapsto & ok(s) \\ x & \mapsto & x \end{array}\right.$$

$$[\![l :]\!] = \left|\begin{array}{lcl} goto(s,l) & \mapsto & ok(s) \\ x & \mapsto & x \end{array}\right.$$

$$[\![\texttt{if}(bx)\ st_1\ \texttt{else}\ st_2]\!] = \left|\begin{array}{lcl} ok(s) & \mapsto & \begin{cases} ok(s') & \text{if } [\![bx]\!](s) = true \quad \wedge \\ & \quad [\![st_1]\!](ok\ s) = ok(s') \\ [\![st_2]\!] \circ [\![st_1]\!](ok\ s) & \text{if } [\![bx]\!](s) = true \quad \wedge \\ & \quad [\![st_1]\!](ok\ s) \neq ok(-) \\ [\![st_2]\!](ok\ s) & \text{if } [\![bx]\!](s) = false \end{cases} \\ x & \mapsto & \begin{cases} ok(s') & \text{if } [\![st_1]\!](x) = ok(s') \\ [\![st_2]\!] \circ [\![st_1]\!](x) & \text{otherwise} \end{cases} \end{array}\right.$$

$$
\begin{array}{rcl}
\mathit{labels} & : & \textsf{\textit{labeled-statement}} \longrightarrow \mathcal{P}(\mathbb{Z}) \\
\mathit{labels}(st) & = & \textsf{collect all case labels in } \textsf{\textit{st}} \textsf{ not covered by an inner switch} \\
sw(bo, x) & : & (SRes \Rightarrow SRes) \times SRes \longrightarrow SRes \\
sw(bo, x) & = & \begin{cases} ok(s) & \textsf{if } bo(x) = break(s) \\ ok(s) & \textsf{if } bo(x) = default(s) \\ bo(x) & \textsf{otherwise} \end{cases}
\end{array}
$$

$$
[\![\mathtt{switch}(ix)\ st]\!] = \left|\begin{array}{rcl}
break(s) & \mapsto & break(s) \\
default(s) & \mapsto & default(s) \\
case(s, i) & \mapsto & case(s, i) \\
ok(s) & \mapsto & \begin{cases} sw([\![st]\!], case(s, i)) & \textsf{if } [\![ix]\!] = ok(i) \wedge i \in labels(st) \\ sw([\![st]\!], default(s)) & \textsf{if } [\![ix]\!] = ok(i) \wedge i \notin labels(st) \\ fail & \textsf{otherwise} \end{cases} \\
x & \mapsto & \begin{cases} ok(s) & \textsf{if } [\![st]\!](x) = break(s) \\ [\![st]\!](x) & \textsf{otherwise} \end{cases}
\end{array}\right.
$$

$$iter \quad : \quad (S \to bool) \times (Result \to Result) \times Result \longrightarrow \mathbb{N} \longrightarrow Result \times \mathbb{B}$$

$$iter(co, bo, x) \quad = \quad \left|\begin{array}{lll} 0 & \mapsto & (x, true) \\ n & \mapsto & \begin{cases} (x, false) & \text{if } x = ok(s) \;\wedge\; co(s) = false \\ iter(co, bo, bo(x))\ (n-1) & \text{otherwise} \end{cases} \end{array}\right.$$

$$term? \quad : \quad Result \times bool \longrightarrow bool$$

$$term? \quad = \quad \left|\begin{array}{lll} (ok(s), false) & \mapsto & true \\ (ok(s), true) & \mapsto & false \\ (-,-) & \mapsto & true \end{array}\right.$$

$$hangs? \quad : \quad (S \to bool) \times (Result \to Result) \times Result \longrightarrow \mathbb{B}$$

$$hangs?(co, bo, x) \quad = \quad \begin{cases} true & \text{if } \forall i \in \mathbb{N}^+ \,.\, term?(iter(co, bo, i)) = false \\ false & \text{otherwise} \end{cases}$$

$$while \quad : \quad (S \to bool) \times (Result \to Result) \times Result \longrightarrow Result$$

$$while(co, bo, x) \quad = \quad \begin{cases} hang & \text{if } hangs?(co, bo, x) = true \\ \pi_1(iter(co, bo, x)\ (n)) & \text{otherwise} \\ \quad \text{where } n = \min\{i \in \mathbb{N}^+ \mid term?(iter(co, bo, x)\ (i))\} & \end{cases}$$

$$[\![\texttt{while}(bx)\ st]\!] \quad = \quad \left|\begin{array}{lll} break(s) & \mapsto & break(s) \\ x & \mapsto & \begin{cases} ok(s) & \text{if } while([\![bx]\!], [\![st]\!], x) = break(s) \\ while([\![bx]\!], [\![st]\!], x) & \text{otherwise} \end{cases} \end{array}\right.$$

$$g\_iter \quad : \quad (Result \rightarrow Result) \longrightarrow \mathbb{N} \longrightarrow Result \longrightarrow Result$$

$$g\_iter\ bo \quad = \quad \left| \begin{array}{lll} 0 & \mapsto & bo \\ n & \mapsto & bo \circ (g\_iter\ bo\ (n-1)) \end{array} \right.$$

$$g\_term? \quad : \quad Result \longrightarrow bool$$

$$g\_term? \quad = \quad \left| \begin{array}{ccl} goto(s) & \mapsto & false \\ x & \mapsto & true \end{array} \right.$$

$$g\_hangs? \quad : \quad (Result \rightarrow Result) \times Result \longrightarrow bool$$

$$g\_hangs?(bo, x) \quad = \quad \begin{cases} true & \text{if } \forall i \in \mathbb{N} \,.\, g\_term?(g\_iter\ bo\ i\ x) = false \\ false & \text{otherwise} \end{cases}$$

$$[\![block]\!] \quad : \quad S \longrightarrow Result$$

$$[\![block]\!](s) \quad = \quad \begin{cases} hang & \text{if } g\_hangs?([\![block]\!], ok(s)) = true \\ g\_iter\ [\![block]\!]\ n\ (ok\ s) & \text{otherwise} \\ \quad \text{where } n = \min\{i \in \mathbb{N} \mid g\_term?(g\_iter\ [\![block]\!]\ i\ (ok\ s))\} & \end{cases}$$

Standard Hoare calculus

$$\frac{\{P \wedge co\} \quad bo \quad \{P\}}{\{P\} \quad \textbf{while}(co) \; bo \quad \{P \wedge \neg co\}}$$

In a real language with abrupt termination and side effects

- the loop might terminate *although* the condition is true
- the condition might be true *just after* it has been evaluated to false

In a hackers language with jumps into loops

- might enter the loop without evaluating the condition

Definition 1 (Goto-While Invariant) *A pair of predicates* $P, Q \subseteq Result$ *is a* goto-while invariant *with respect to the condition expression co and the loop body st if for all* $x \in Result$

$$P(x) \quad \textit{implies} \quad \begin{cases} Q(\llbracket co \rrbracket(x)) & \textit{if } \textit{n-term}(co, x) \\ P(ok(s)) & \textit{else if } \llbracket st \rrbracket \circ \llbracket co \rrbracket(x) = ok(s) \\ Q(\llbracket st \rrbracket \circ \llbracket co \rrbracket(x)) & \textit{otherwise} \end{cases}$$

where the "else if" case is effective only if n-term$(co, x) = false$.

n-term checks whether the loops terminates in the traditional way via a false condition:

$$\textit{n-term}(co, x) \stackrel{\textit{def}}{=} \begin{cases} true & \textit{if } x = ok(s) \wedge \llbracket co \rrbracket(s) = ok(false) \\ false & \textit{otherwise} \end{cases}$$

Definition 2 (Goto-While Variant) *Let*

- *condition co and body st as before*
- $(O, <)$ *be a well-founded order*
- f *be a function* $Result \longrightarrow O$
- (P, Q) *be a goto-while invariant*

The function f *is a* goto-while variant *just in case* $\forall x \in Result$:

$$P(x) \wedge \textit{n-term}(co, x) = false \wedge \llbracket st \rrbracket \circ \llbracket co \rrbracket(x) = ok(-) \implies f(\llbracket st \rrbracket \circ \llbracket co \rrbracket(x)) < f(x)$$

Theorem 3 (Total Goto-While Correctness) *Consider*

- *condition co and body st*
- *goto-while invariant* (P, Q)
- *variant* f

For all $x \in \mathit{Result}$ *with* $x \neq \mathit{break}(-)$ *:*

$$P(x) \quad \textit{implies} \quad Q^\dagger\big([\![\texttt{while}(co)\ st]\!](x)\big)$$

The dagger † deals with catching breaks at the end of while loops:

$$Q^\dagger \overset{\mathit{def}}{=} \{\, x \in \mathit{Result} \mid (x \neq \mathit{break}(-) \wedge Q(x)) \vee \exists s \in \mathit{State} \,.\, x = \mathit{ok}(s) \wedge Q(\mathit{break}\ s) \,\}$$

$$Q^\dagger \approx \coprod_{\mathit{catch\_break}} Q$$

Translate source code into PVS

```
rounds : posnat = 1
i : posnat = 2

duff(source, dest : posnat, count : nat) : [Result[State, Unit] -> Result[State,Unit]] =
  write_int(rounds, div(const_int(count), const_int(8))) ##
  write_int(i, const_int(0)) ##
  int_switch_stm( rem( const_int(count), const_int(8)),
      int_case(0) ##
              gwhile_stm(
                 const_int(0) < post_decr_const(rounds),
                      write_int_array(dest, read_int(i), read_int_array(source, read_int(i))) ##
                      write_int(i, read_int(i) ++ const_int(1)) ##
      int_case(7) ##  write_int_array(dest, read_int(i), read_int_array(source, read_int(i))) ##
                      write_int(i, read_int(i) ++ const_int(1)) ##
      int_case(6) ##  write_int_array(dest, read_int(i), read_int_array(source, read_int(i))) ##
                      write_int(i, read_int(i) ++ const_int(1)) ##
          .................
      int_case(2) ##  write_int_array(dest, read_int(i), read_int_array(source, read_int(i))) ##
                      write_int(i, read_int(i) ++ const_int(1)) ##
      int_case(1) ##  write_int_array(dest, read_int(i), read_int_array(source, read_int(i))) ##
                      write_int(i, read_int(i) ++ const_int(1))
              )  % end gwhile_stm
  ) % end int_switch_stm
```

Proof it

```
duff_total : Lemma
  Forall(s : State, source, dest : posnat, count : nat) :
    duff_var_ok(source, dest, count,s) Implies
      duff(source, dest, count)(ok(s,unit)) =
      ok(s WITH [
          `vars := Lambda(j : posnat) :
              IF j = rounds Then int(−1)
              Elsif j = i Then int(count)
              Elsif cell_in_array(s, dest)(j) And
                      index_from_cell(s, dest, j) < count
                  Then
                      s`vars(get_array(s`vars(source))
                              `fields(index_from_cell(s, dest, j)))
              Else s`vars(j)
              Endif
         ], unit)
```

Working Semantics for “dirty” programs

- abrupt termination
- goto, switch
- simple compositional denotational semantics (no continuations, no domain theory)
- semantics is equivalent with traditional least fixpoint definition
- Hoare rule for while treating all abnormalities and even jumps into the loop
- Duff's device verified