Input, Output, and Automation in x86 Proved

Jason Gross

Hosted by Andrew Kennedy

Summer 2014

Verified I/O Programs

> "When doing formal verification, come up with the simplest non-trivial example you can. Then start with something simpler."
>
> —Adam Chlipala (paraphrased)

Precondition

Postcondition

```
{true} while true do skip done {false}
[]
```

I/O behavior

Verified I/O Programs: Trivial Loop

`{true} while true do skip done {false}`

`[]`

```
Example safe_loop_while eax :
 ⊢ basic (EAX ≅ eax ⋆ OSZCP?) (while (TEST EAX, EAX) CC_O false prog_skip) [] false.
Proof.
 basic apply (while_rule_ro (I := λ b ⇒ b = false ∧ EAX? ⋆ SF? ⋆ ZF? ⋆ CF? ⋆ PF?)) ⇒ //=;
  rewrite /stateIsAny; specintros ⇒ *;
  basic apply *.
Qed.
```

Verified I/O Programs: Trivial Loop

```
{true} while true do skip done {false}
```

[]

Precondition

Code

Postcondition

```
Example safe_loop_while eax :
 ⊢ basic (EAX ≅ eax ⋆ OSZCP?) (while (TEST EAX, EAX) CC_O false prog_skip) [] false.
Proof.
 basic apply (while_rule_ro (I := λ b ⇒ b = false ∧ EAX? ⋆ SF? ⋆ ZF? ⋆ CF? ⋆ PF?)) ⇒ //=;
  rewrite /stateIsAny; specintros ⇒ *;
  basic apply *.
Qed.
```

Loop invariant

Verified I/O Programs: Eternal Output

`{true} while true do out 1 done {false}`

`1*`

Kleene star

```
Example loop_forever_one channel :
 ⊢ basic (AL?)
          (MOV AL, 1;;
           LOCAL LOOP;
           LOOP:;;
           OUT channel, AL;;
           JMP LOOP)
          1*
          false.
Proof.
 (elided)
Qed.
```

Verified I/O Programs: Eternal Output

{true} **while** true **do out** 1 **done** {false}

1*

```
Example loop_forever_one channel :
  ⊢ loopy_basic (AL?)
                (MOV AL, (#1 : DWORD);;
                 LOCAL LOOP;
                 LOOP:;;
                 OUT channel, AL;;
                 JMP LOOP)
                (starOP (outOP (zeroExtend n8 channel) (#1 : BYTE)))
                lfalse.
Proof.
  (still elided)
Qed.
```

Verified I/O Programs: Echo Once

∀ v, {true}   v ← **input; out** v   {true}

In v; Out v

```
Example safe_echo_once in_channel out_channel :
 ⊢ ∀ v, basic (AL?)
              (IN in_channel, AL;;
               OUT out_channel, AL)
              [In v; Out v]
              (AL ≅ v).
Proof.
 rewrite /stateIsAny; specintros ⇒ al v.
 basic apply ∗.
 basic apply ∗.
Qed.
```

Digression: Hoare Rule for While

$$\frac{\{P \wedge B\}\ S\ \{P\}}{\{P\}\ \textbf{while}\ B\ \textbf{do}\ S\ \textbf{done}\ \{\ \neg B \wedge P\ \}}$$

Standard rule from Wikipedia

$$\frac{\overbrace{\{P \wedge B\}\ S\ \{P\}}^{O_S} \qquad \to (O_S + O' \vdash O) \quad \to ([] \vdash O)}{\underbrace{\{P\}\ \textbf{while}\ B\ \textbf{do}\ S\ \textbf{done}\ \{\ \neg B \wedge P\ \}}_{O}}$$

The I/O doesn't get to talk about state!

What's O'?

Desired output annotations

Digression: Hoare Rule for While

$$\frac{\{P \wedge B\}\ S\ \{P\}}{\{P\}\ \textbf{while}\ B\ \textbf{do}\ S\ \textbf{done}\ \{\ \neg B \wedge P\ \}}$$

I lets us "negate" B, (which is code)

The test of B had value v

$$\frac{\{P\}\ \text{test}\ B\ \{\exists v, I\ v \star B{\sim}v\} \qquad \{I\ \text{true} \star B{\sim}\text{true}\}\ S\ \{P\}}{\{P\}\ \textbf{while}\ B\ \textbf{do}\ S\ \textbf{done}\ \{\ I\ \text{false} \star B{\sim}\text{false}\}}$$

Digression: Hoare Rule for While Example

$$\{P\}$$

$$\textbf{while}\ (x > 0)\ \textbf{do}\ (\mathrm{OUT}\ x; x{-}{-})\ \textbf{done}$$

$$\{\ I\ \text{false} \star B{\sim}\text{false}\}$$

We can drop B, which is technical info about flags

$$\frac{\{P\}\ \text{test}\ B\ \{\exists v, I\ v \star B{\sim}v\} \quad \{I\ \text{true} \star B{\sim}\text{true}\}\ S\ \{P\}}{\{P\}\ \textbf{while}\ B\ \textbf{do}\ S\ \textbf{done}\ \{\ I\ \text{false} \star B{\sim}\text{false}\}}$$

Digression: Hoare Rule for While Example

$$\{P\}\ \textbf{while}\ (x > 0)\ \textbf{do}\ (\text{OUT}\ x; x{-}{-})\ \textbf{done}\ \{\ I\ \text{false}\}$$

$$\frac{\{P\}\ \text{test}\ B\ \{\exists v, I\ v\} \qquad \{I\ \text{true}\}\ S\ \{P\}}{\{P\}\ \textbf{while}\ B\ \textbf{do}\ S\ \textbf{done}\ \{\ I\ \text{false}\}}$$

Digression: Hoare Rule for While Example

$$\frac{\overline{\{\text{true}\}\ \text{test}\ (x > 0)\ \{\exists v, (x > 0) = v\}} \qquad \overline{\{x > 0\}\ (\text{OUT}\ x; x{-}{-})\ \{\text{true}\}}}{\{\text{true}\}\ \textbf{while}\ (x > 0)\ \textbf{do}\ (\text{OUT}\ x; x{-}{-})\ \textbf{done}\ \{\ x \leq 0\}}$$

$$\frac{\{P\}\ \text{test}\ B\ \{\exists v, I\ v\} \qquad \{I\ \text{true}\}\ S\ \{P\}}{\{P\}\ \textbf{while}\ B\ \textbf{do}\ S\ \textbf{done}\ \{\ I\ \text{false}\}}$$

Digression: Hoare Rule for While Example with output

$$\frac{\overline{\{\text{true}\}\ \text{test}\ (x > 0)\ \{\exists v, (x > 0) = v\}} \qquad \overline{\{x > 0\}\ (\text{OUT}\ x; x{-}{-})\ \{\text{true}}}{\underbrace{\{x = n\}\ \textbf{while}\ (x > 0)\ \textbf{do}\ (\text{OUT}\ x; x{-}{-})\ \textbf{done}\ \{\ x \leq 0\}}_{[n,n-1,n-2,\dots,1]}}$$

$$\frac{\overbrace{\{P\}\ \text{test}\ B\ \{\exists v, I\ v\}}^{[]} \qquad \overbrace{\{I\ \text{true}\}\ S\ \{P\}}^{o_S}}{\underbrace{\{P\}\ \textbf{while}\ B\ \textbf{do}\ S\ \textbf{done}\ \{\ I\ \text{false}\}}_{o}}$$

Worry about side conditions later

Digression: Hoare Rule for While Example with output

Uh-oh!

$$\frac{\{x = n\}\ \text{test}\ (x > 0)\ \{\exists v, (x > 0) = v\} \qquad \{x > 0\}\ (\text{OUT}\ x; x--)\ \{x = n}{\underbrace{\{x = n\}\ \textbf{while}\ (x > 0)\ \textbf{do}\ (\text{OUT}\ x; x--)\ \textbf{done}\ \{\ x \leq 0\}}_{[n, n-1, n-2, \ldots, 1]}}$$

$$\frac{\overbrace{\{P\}\ \text{test}\ B\ \{\exists v, I\ v\}}^{[]} \qquad \overbrace{\{I\ \text{true}\}\ S\ \{P\}}^{o_S}}{\underbrace{\{P\}\ \textbf{while}\ B\ \textbf{do}\ S\ \textbf{done}\ \{\ I\ \text{false}\}}_{o}}$$

Worry about side conditions later

Digression: Hoare Rule for While

$$\frac{\{P \land B\}\ S\ \{P\}}{\{P\}\ \textbf{while}\ B\ \textbf{do}\ S\ \textbf{done}\ \{\ \neg B \land P\ \}}$$

Eliding $B \sim v$ for space

"transition function"

$$\frac{\forall x, \{I\ x\ \text{true}\}\ S\ \{P\ (\mathrm{T}\ x)\} \qquad \forall x, I\ x\ \text{true} \land Q\ (\mathrm{T}\ x) \vdash Q\ x \\ \forall x, \{P\ x\}\ \text{test}\ B\ \{\exists v, I\ x\ v\} \qquad \forall x, I\ x\ \text{false} \vdash Q\ x}{\forall x, \{P\ x\}\ \textbf{while}\ B\ \textbf{do}\ S\ \textbf{done}\ \{Q\ x\}}$$

Digression: Hoare Rule for While

$$\frac{\{P \land B\}\ S\ \{P\}}{\{P\}\ \textbf{while}\ B\ \textbf{do}\ S\ \textbf{done}\ \{\ \neg B \land P\ \}}$$

Can't talk about state here

$$\frac{\begin{array}{cc} \forall x, \underbrace{\{I\ x\ \text{true}\}\ S\ \{P\ (\mathrm{T}\ x)\}}_{O_S\ x} & \forall\ x, I\ x\ \text{true} \to O_S\ x + O\ (T\ x) \vdash O\ x \\ & \forall\ x, I\ x\ \text{false} \to [] \vdash O\ x \\ \forall x, \underbrace{\{P\ x\}\ \text{test}\ B\ \{\exists v, I\ x\ v\}}_{[]} & \forall x, I\ x\ \text{true} \land Q\ (\mathrm{T}\ x) \vdash Q\ x \\ & \forall x, I\ x\ \text{false} \vdash Q\ x \end{array}}{\forall x, \underbrace{\{P\ x\}\ \textbf{while}\ B\ \textbf{do}\ S\ \textbf{done}\ \{Q\ x\}}_{O\ x}}$$

Digression: Hoare Rule for While

$$\frac{\{P \land B\}\; S\; \{P\}}{\{P\}\;\textbf{while}\; B\; \textbf{do}\; S\; \textbf{done}\; \{\, \neg B \land P\, \}}$$

Logical part of I

$$\frac{\begin{array}{ll} \forall x, \underbrace{\{I\ x\ \text{true}\}\; S\; \{P\ (\mathrm{T}\ x)\}}_{O_S\ x} & \forall\, x, I_L\ x\ \text{true} \rightarrow O_S\ x + O\ (T\ x) \vdash O\ x \\ & \forall\, x, I_L\ x\ \text{false} \rightarrow [] \vdash O\ x \\ \forall x, \underbrace{\{P\ x\}\; \text{test}\; B\; \{\exists v, I\ x\ v\}}_{[]} & \forall x, I\ x\ \text{true} \land Q\ (\mathrm{T}\ x) \vdash Q\ x \\ & \forall x, I\ x\ \text{false} \vdash Q\ x \end{array}}{\forall x, \underbrace{\{P\ x\}\; \textbf{while}\; B\; \textbf{do}\; S\; \textbf{done}\; \{Q\ x\}}_{O\ x}}$$

Digression: Hoare Rule for While Example with output

$\forall n, \underbrace{\{n > 0 \wedge x = n\}\ (\mathrm{OUT}\ x; x{-}{-})\ \{x = n - 1\}}_{\mathrm{OUT}\ n}$

$\forall n, n > 0 \rightarrow \mathrm{OUT}\ n + O\ (n-1) \vdash O\ n$

$\ldots \leq 0 \rightarrow [\,] \vdash O\ n$

$\forall n, \underbrace{\{x = n\}\ \mathrm{test}\ (x > 0)\ \{\exists v, (n > 0) \ldots}_{[\,]}$

$\ldots n \wedge x \leq 0 \vdash x \leq 0$

$\ldots \vdash x \leq 0$

$\forall n, \underbrace{\{x = n\}\ \mathbf{while}\ (\ldots \mathbf{done}\ \{x \leq 0\}}_{O\ n}$

On-the-fly demo at the end, time and interest permitting

$\forall x, \underbrace{\{I\ x\ \mathrm{true}\}\ S\ \{P\ (\mathrm{T}\ x)\}}_{O_S\ x}$

$\forall x, \underbrace{\{P\ x\}\ \mathrm{test}\ B\ \{\exists v, I\ x\ v\}}_{[\,]}$

$\forall x, I_L\ x\ \mathrm{true} \rightarrow O_S\ x + O\ (T\ x) \vdash O\ x$

$\forall x, I_L\ x\ \mathrm{false} \rightarrow [\,] \vdash O\ x$

$\forall x, I\ x\ \mathrm{true} \wedge Q\ (\mathrm{T}\ x) \vdash Q\ x$

$\forall x, I\ x\ \mathrm{false} \vdash Q\ x$

$$\forall x, \underbrace{\{P\ x\}\ \mathbf{while}\ B\ \mathbf{do}\ S\ \mathbf{done}\ \{Q\ x\}}_{O\ x}$$

Verified I/O Programs: Echo

∀ *vs* : stream,

{true}**while** true **do** (v ← **input**; **out** v) **done**{true}

map (*v* ↦ In *v*; Out *v*) *vs*

```
Example safe_echo eax in_channel out_channel :
 ⊢ ∀ vs, basic  (AL? ⋆ EAX ≅ eax ⋆ OSZCP?)
                (while (TEST EAX, EAX) CC_O false (
                          IN in_channel, AL;;
                          OUT out_channel, AL
                )
                (stream_Opred_map (λ v ↦ [In v; Out v]) vs)
                lfalse.
Proof.
 (elided; 8 lines of filling in arguments to the while rule, 9 lines of automation about specs)
Qed.
```

Verified I/O Programs: Accumulator

∀ vs : list (non-zero BYTE),

```
{acc=x}
 while ((v ← input) ≠ 0) do (acc = accumulate acc v) done
                                     {acc = fold accumulate x vs}
                              map (v ↦ In v) vs
```

```
Example addB_until_zero_prog_safe ch o s z c p S al
: S ⊢ (∀ initial (x : BYTE) (xs : seq BYTE) (pf1 : only_last (λ t : BYTE ⇒ t == #0) x xs),
     (loopy_basic (AH ≅ initial ⋆ AL ≅ al ⋆ OSZCP o s z c p)
                  ( IN ch, AL;;
                    while (CMP AL, #0) CC_Z false (ADD AH, AL;; IN ch, AL))
                  (foldr catOP empOP (map (inOP (zeroExtend 8 ch)) (x::xs)))
                  ((AH ≅ (foldl addB initial (drop_last x xs)))
                          ⋆ AL ≅ #0 ⋆ OF? ⋆ SF? ⋆ ZF ≅ true ⋆ CF? ⋆ PF?))).
Proof.
 specintros ⇒ *.
 basic apply (@accumulate_until_zero_prog_safe _ (λ x ⇒ AH ≅ x)) ⇒ *; first assumption.
 basic apply *.
Qed.
```

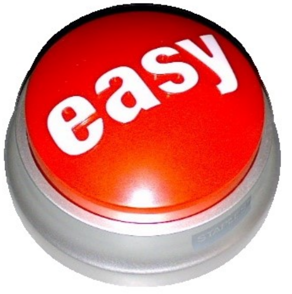
Verified I/O Programs: Next Steps

- readline (via accumulator template)
- prime number printer
- text adventure?
- use memory-mapped I/O rather than IN and OUT

Automation

Instruction Automation: Ideal

1. Define the instruction in the model
2. State the high-level (e
3. Push-button verificati



Maybe even omit 2, if the
good enough.

Image from https://flic.kr/p/3KYpfJ, "State of Mind", **tshein**

Instruction Automation: Reality

1. Define the instruction in the model

```
Definition evalBinOp {n} op : BITS n → BITS n → ST (BITS n) :=
 match op with
  | OP_XOR ⇒ evalLogicalOp xorB
  ...
 end.
Definition evalLogicalOp {n} (f : BITS n → BITS n → BITS n) arg1 arg2 :=
 let result := f arg1 arg2 in
 do! updateFlagInProcState CF false;
 do! updateFlagInProcState OF false;
 do! updateZPS result;
 retn result.
```

Instruction Automation: Reality

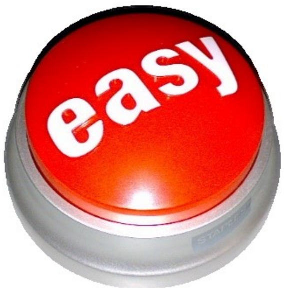
2. State the high-level (e.g., Hoare) rule

```
Lemma XOR_RR_rule s (r1 r2:VReg s) v1 (v2:VWORD s):
 ⊢ basic   (VRegIs r1 v1 ⋆ VRegIs r2 v2 ⋆ OSZCP?)
           (XOR r1, r2)
           []
           (VRegIs r1 (xorB v1 v2) ⋆ VRegIs r2 v2
                  ⋆ OSZCP false (msb (xorB v1 v2))
                          (xorB v1 v2 == #0) false (lsb (xorB v1 v2))).
```

Instruction Automation: Reality

3. Push-button verification

```
Lemma XOR_RR_rule s (r1 r2:VReg s) v1 (v2:VWORD s):
 ⊢ basic   (VRegIs r1 v1 ⋆ VRegIs r2 v2 ⋆ OSZCP?)
           (XOR r1, r2)
           []
           (VRegIs r1 (xorB v1 v2) ⋆ VRegIs r2 v2
                  ⋆ OSZCP false (msb (xorB v1 v2))
                             (xorB v1 v2 == #0) false (lsb (xorB v1 v2))).
Proof. destruct s; do_instrrule_triple. Qed.
```



Instruction Automation: Old Reality

3. Push-button verification

File Edit Options Buffers Tools Coq Proof-General Holes Help

```
Lemma XOR_RR_rule (r1 r2:Reg) v1 (v2:DWORD):
 |-- basic (r1~=v1 ** r2~=v2 ** OSZCP_Any) (XOR r1, r2)
           (r1~=xorB v1 v2 ** r2~=v2 ** OSZCP false (msb (xorB v1 v2))
                           (xorB v1 v2 == #0) false (lsb (xorB v1 v2))).
Proof.
 apply TRIPLE_basic => R.
 repeat autounfold with eval.
 triple_apply evalReg_rule.
 triple_apply evalReg_rule.
 triple_apply triple_pre_introFlags => o s z c p. rewrite /OSZCP/evalLogicalOp.
 triple_apply triple_doSetFlagSep.
 triple_apply triple_doSetFlagSep.
 triple_apply triple_doUpdateZPS.
 triple_apply triple_setRegSep.
Qed.
```

Instruction Automation: Old Reality

emacs@MSRC-3617289

File Edit Options Buffers Tools Coq Proof-General Holes Help

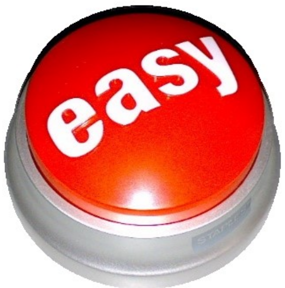
```
Lemma ADDSUB_rule isSUB d (ds:DstSrc d) v1 :
  |-- specAtDstSrc ds (fun D v2 =>
      basic (D v1 ** OSZCP_Any)
            (BOP d (if isSUB then OP_SUB else OP_ADD) ds)
            (let: (carry,v) := (if isSUB then sbbB else adcB) false v1 v2 in
             D v ** OSZCP (computeOverflow v1 v2 v) (msb v) (v == #0) carry (lsb v))).
Proof.
 rewrite /specAtDstSrc.
 destruct ds.
 (* RR *)
 specintros => v2.
 autorewrite with push_at. apply TRIPLE_basic => R. rewrite /evalInstr/evalDstSrc/evalDstR.
 triple_apply evalDWORDorBYTEReg_rule.
 triple_apply evalDWORDorBYTEReg_rule.
 rewrite /evalBinOp/evalArithOpNoCarry.
 triple_apply triple_pre_introFlags => o s z c pf. rewrite /OSZCP.
 destruct isSUB;
   (elim: (_ false v1 v2) => [carry v];
   triple_apply triple_doSetFlagSep;
   triple_apply triple_doSetFlagSep;
   triple_apply triple_doUpdateZPS;
   triple_apply triple_setDWORDorBYTERegSep).
 (* RM *)
 rewrite /specAtMemSpec.
 elim:src => [optSIB offset].
 elim: optSIB => [[base ixopt] |].
 case: ixopt => [[ixr sc] |].
(* Indexed *)
 + specintros => v2 pbase ixval.
   autorewrite with push_at. apply TRIPLE_basic => R. rewrite /evalInstr/evalDstSrc/evalDstR.
   triple_apply evalDWORDorBYTEReg_rule.
   triple_apply evalMemSpec_rule.
   triple_apply triple_letGetDWORDorBYTESep.
   rewrite /evalBinOp/evalArithOpNoCarry.
   triple_apply triple_pre_introFlags => o s z c pf. rewrite /OSZCP.
 destruct isSUB;
   (elim: (_ false v1 v2) => [carry v];
   triple_apply triple_doSetFlagSep;
   triple_apply triple_doSetFlagSep;
   triple_apply triple_doUpdateZPS;
   triple_apply triple_setDWORDorBYTERegSep).
(* Non-indexed *)
 + specintros => v2 pbase.
   autorewrite with push_at. apply TRIPLE_basic => R. rewrite /evalInstr/evalDstSrc/evalDstR.
   triple_apply evalDWORDorBYTEReg_rule.
   triple_apply evalMemSpecNone_rule.
   triple_apply triple_pre_introFlags => o s z c pf. rewrite /OSZCP.
   triple_apply triple_letGetDWORDorBYTESep.
   rewrite /evalBinOp/evalArithOpNoCarry.
 destruct isSUB;
   (elim: (_ false v1 v2) => [carry v];
   triple_apply triple_doSetFlagSep;
   triple_apply triple_doSetFlagSep;
   triple_apply triple_doUpdateZPS;
   triple_apply triple_setDWORDorBYTERegSep).
(* offset only *)
 + specintro => v2.
   autorewrite with push_at. apply TRIPLE_basic => R. rewrite /evalInstr/evalDstSrc/evalDstR.
   triple_apply evalDWORDorBYTEReg_rule.  rewrite /evalMemSpec.
   triple_apply triple_letGetDWORDorBYTESep.
   triple_apply triple_pre_introFlags => o s z c pf. rewrite /OSZCP.
   rewrite /evalBinOp/evalArithOpNoCarry.
 destruct isSUB;
   (elim: (_ false v1 v2) => [carry v];
```

∏U:**- foo.v   Top L1   (Coq Holes)

Instruction Automation: Mechanics

```
Lemma XOR_RR_rule s (r1 r2:VReg s) v1 (v2:VWORD s):
 ⊢ basic   (VRegIs r1 v1 ⋆ VRegIs r2 v2 ⋆ OSZCP?)
           (XOR r1, r2)
           []
           (VRegIs r1 (xorB v1 v2) ⋆ VRegIs r2 v2
                 ⋆ OSZCP false (msb (xorB v1 v2))
                           (xorB v1 v2 == #0) false (lsb (xorB v1 v2))).
Proof. destruct s; do_instrrule_triple. Qed.
```

1. Lookup
2. Application
3. Heuristics



Instruction Automation: Mechanics

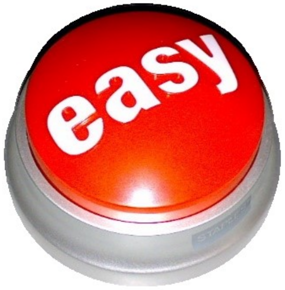
Automated timing scripts helped ensure that the automation didn't slow things down unreasonably.

Example diff

After	File Name	Before	Change
17m33.61s	Total	18m51.54s	-1m17.92s
0m35.85s	examples/mulc	0m42.18s	-0m06.32s
0m33.42s	examples/specexamples	0m27.19s	+0m06.23s
0m24.29s	x86/lifeimp	0m30.41s	-0m06.12s
0m17.60s	x86/inlinealloc	0m21.79s	-0m04.18s
0m27.75s	x86/imp	0m31.41s	-0m03.66s
0m17.56s	x86/instrrules/mov	0m20.79s	-0m03.23s
0m15.76s	x86/call	0m19.29s	-0m03.52s
0m51.82s	x86/instrrules/addsub	0m54.54s	-0m02.71s

Program Automation: Ideal

1. Write a program
2. State the spec
3. Sprinkle annotations
4. Push-button verificati



Maybe even omit 3, if the …
good enough.

Image from https://flic.kr/p/3KYpfJ, “State of Mind”, **tshein**

Program Automation: Reality

`specapply *` takes care of essentially all of the unstructured code reasoning

```
Example safe_echo eax in_c out_c :
 ⊢ ∀ vs, basic          ( AL? ⋆ EAX ≅ eax ⋆ OSZCP?)
                        ( while (TEST EAX, EAX) CC_O false (
                                 IN in_c, AL;;
                                 OUT out_c, AL
                        )
                        (eq_opred_stream (stream_to_in_out in_c out_c vs))
                        lfalse.
Proof.
 eapply @while_rule_ind
 with (I_logic := λ _ b ⇒ false == b)
      (Otest := λ _ ⇒ empOP)
      (Obody := λ s ⇒ echo_once_OP_spec in_c out_c (hd s))
      (I_state := λ s _ ⇒ EAX ≅ eax ⋆ AL? ⋆ SF? ⋆ ZF? ⋆ CF? ⋆ PF?)
      (transition_body := @tl _)
      (O_after_test := λ s ⇒ default_PointedOPred
                     (catOP (echo_once_OP_spec in_c out_c (hd s)) (eq_opred_stream (stream_to_in_out in_c out_c (tl s)))));
  do ![ progress rewrite → ?empOPL, → ?eq_opred_stream__echo_once
      | specintros ⇒ *
      | done
      | by ssimpl
      | basic apply *
      | progress simpl OPred_pred
      | progress move ⇒ *
      | progress rewrite /stateIsAny
      | reflexivity ].
Qed.
```

`basic apply *` takes care of *all* of the code reasoning here

Program Automation: Mechanics

1. Lookup (via typeclasses)
2. Application (via helper lemmas)
3. Heuristics (for side conditions)

Open Issues

Open Issues: Pointy Predicates

```
Definition basic P (c : T) (O : OPred) Q : spec :=
  ∀ (i j : DWORD) (O … ,
  (obs O' @ (EIP ≅ j ⋆ Q) → obs (O + O') @ (EIP ≅ i ⋆ P))
      <@ (i -- j ↦ c).
Definition loopy_basic P (c : T) (O … OPred) Q : spec :=
  ∀ (i j : DWORD) (O' : … )),
  (obs O' @ (EIP ≅ j ⋆ Q) → obs (O + O') @ (EIP ≅ i ⋆ P))
      <@ (i -- j ↦ c).
```

Image from https://flic.kr/p/ab1bwS, “**Spiky balls**” Tom Magliery

Open Issues: Quantifier Location

Program Definition obs (O: OPred) := mkspec (λ k P ⇒
∀ (s: ProcState), (P ⋆ ltrue) s → ∃ o, O o ∧
runsForWithPrefixOf k s o) _ _.

Can we hide the quantifiers in the I/O predicate, so that the specification for `echo` doesn't have to quantify over streams?

Currently, both $(\forall\ v, \mathrm{In}\ v; \mathrm{Out}\ v)^*$ and $(\exists\ v, \mathrm{In}\ v; \mathrm{Out}\ v)^*$ say the wrong thing.

Take Home Messages

We can verify the I/O behavior of simple assembly programs.

Putting effort into Coq automation results in tactics which are:

- comparable in speed to manual proofs
- capable of push-button verification in well-specified domains

Acknowledgements

Thanks to:

- Andrew Kennedy, my host here at MSR, and Nick Benton, who is also on this project.
- Georges Gonthier, for help with Coq and SSReflect
- Jonas Jensen, Jesper Bengtson, Gregory Malecha, and Edward Z. Yang for discussions about various aspects of I/O specifications, among other things.

Thanks!

Questions?

Requests for verification demo?