Embedded application designed by ATS language

Kiwamu Okabe @ RIKEN AICS

Remember Heartbleed bug?

Should we use safer language than C?

== In English ==
"Preventing heartbleed bugs with safe programming languages"
http://bluishcoder.co.nz/2014/04/11/preventing-heartbleed-bugs-with-safe-languages.html
== In Japanease ==
"安全なプログラミング言語を使って heartbleed を防ぐには"
https://github.com/jats-ug/translate/blob/master/Web/bluishcoder.co.nz/2014/04/11/preventing-heartbleed-bugs-with-safe-languages.md

"A safer systems programming language could have prevented the bug."

Want the safer language... It's ATS!

☆ http://www.ats-lang.org/

☆ Syntax like ML

☆ DML-style dependent types

☆ Linear types

☆ Optional GC

☆ Optional malloc/free

☆ Optional run-time

Agenda

☆ [1] Demo

☆ [2] ATS language basics

☆ [3] Prop (proof) on ATS

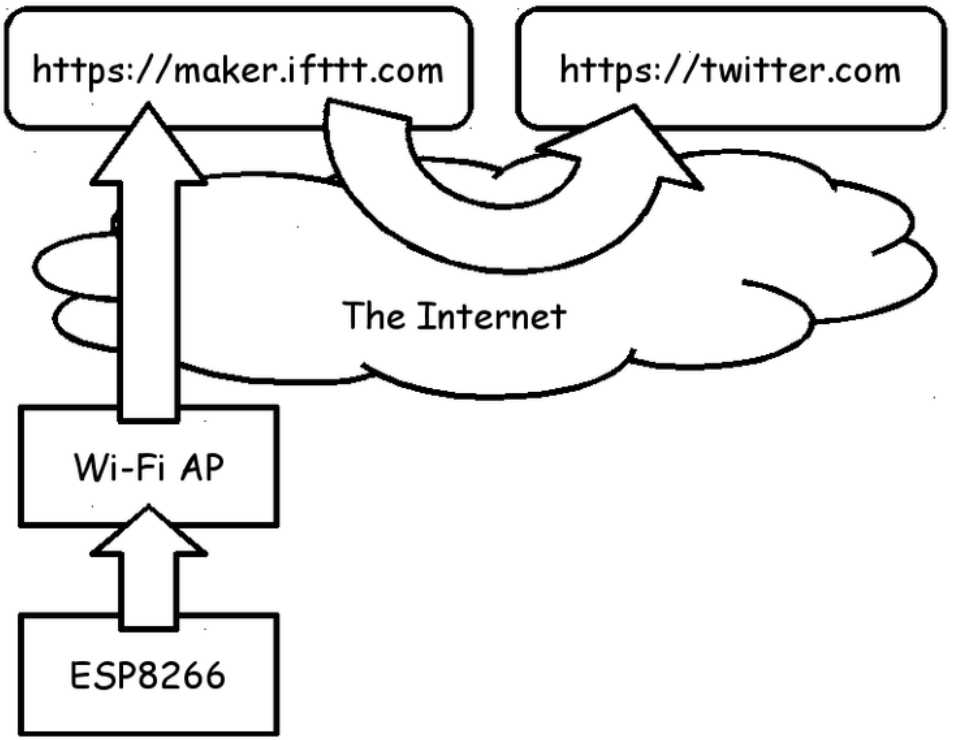
☆ [4] View (linear type) on ATS

☆ [5] Compare ATS with the other

☆ [6] Conclusion

[1] Demo: ATS application on ESP8266

https://github.com/fpiot/esp8266-ats



ESP8266 hardware

☆ http://espressif.com/en/products/wroom/

☆ 32-bit low power MCU Wi-Fi Module

☆ ROM: 4 MB

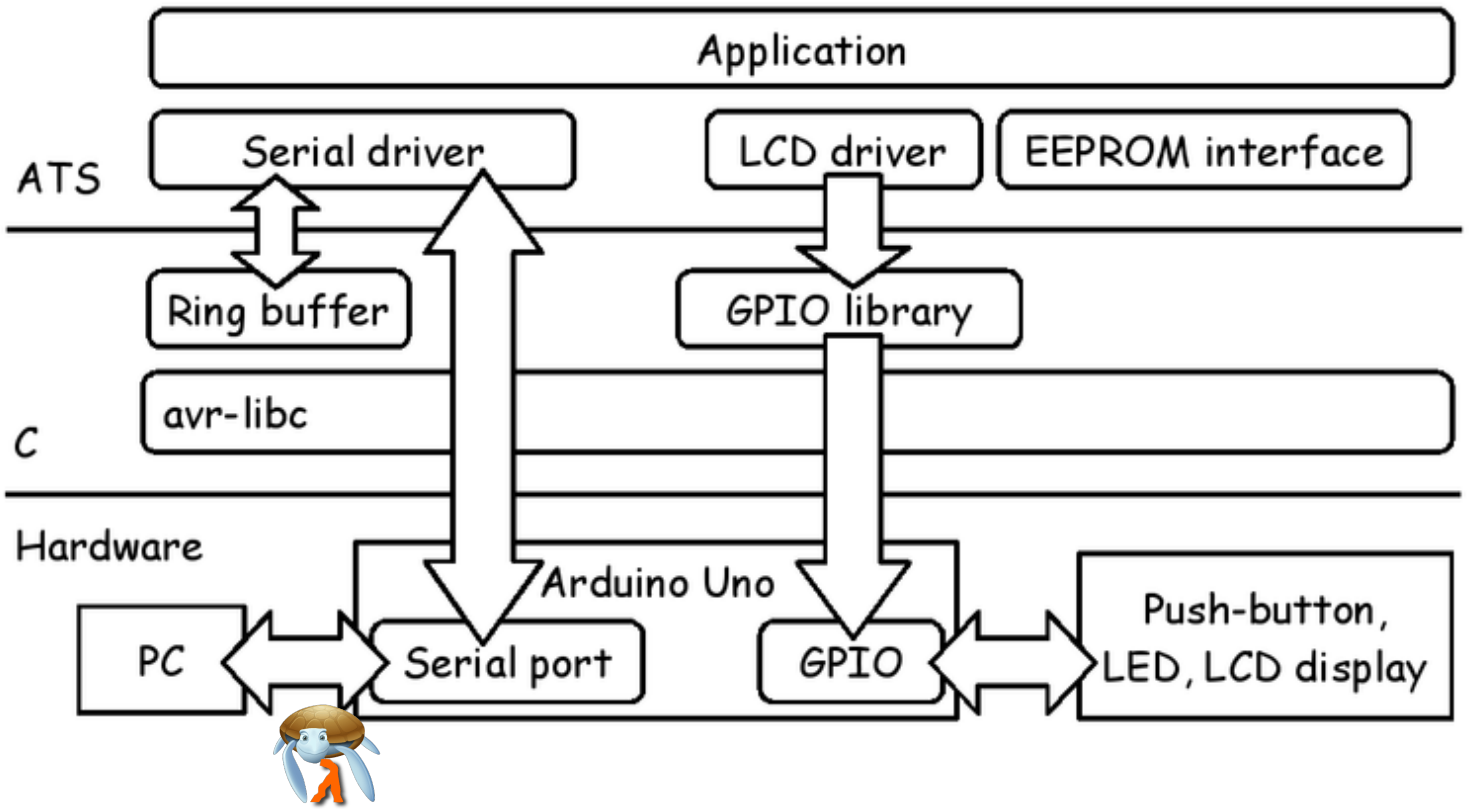
☆ RAM: 50 KB

What error can be captured by ATS?

```
$ vi user/user_main.dats
 39     val json_open = string0_copy "{\"value1\": \""
 40     val json_close = string0_copy "\" }"
 41     val temp = esp_tostrptr_int rand
 42     val json_head = strptr_append (json_open, temp)
 43     val json_data = strptr_append (json_head, json_close)
 44 //  val () = (free json_open; free json_close; free temp; free json_head)
 45     val () = (free json_open; free json_close; free temp) // Error!
$ make
ATS user/user_main.dats
/home/kiwamu/src/esp8266-ats/ifttt_ats/user/user_main.dats: 985(line=32,
offs=43) -- 2237(line=55, offs=4): error(3): the linear dynamic variable
[json_head$3823(-1)] needs to be consumed but it is preserved with the type
[S2Eapp(S2Ecst(strptr_addr_vtype); S2EVar(4441))] instead.
patsopt(TRANS3): there are [1] errors in total.
```


Demo: ATS application on Arduino Uno

https://github.com/fpiot/arduino-ats



Arduino Uno hardware

☆ https://www.arduino.cc/

☆ 8-bit Harvard architecture

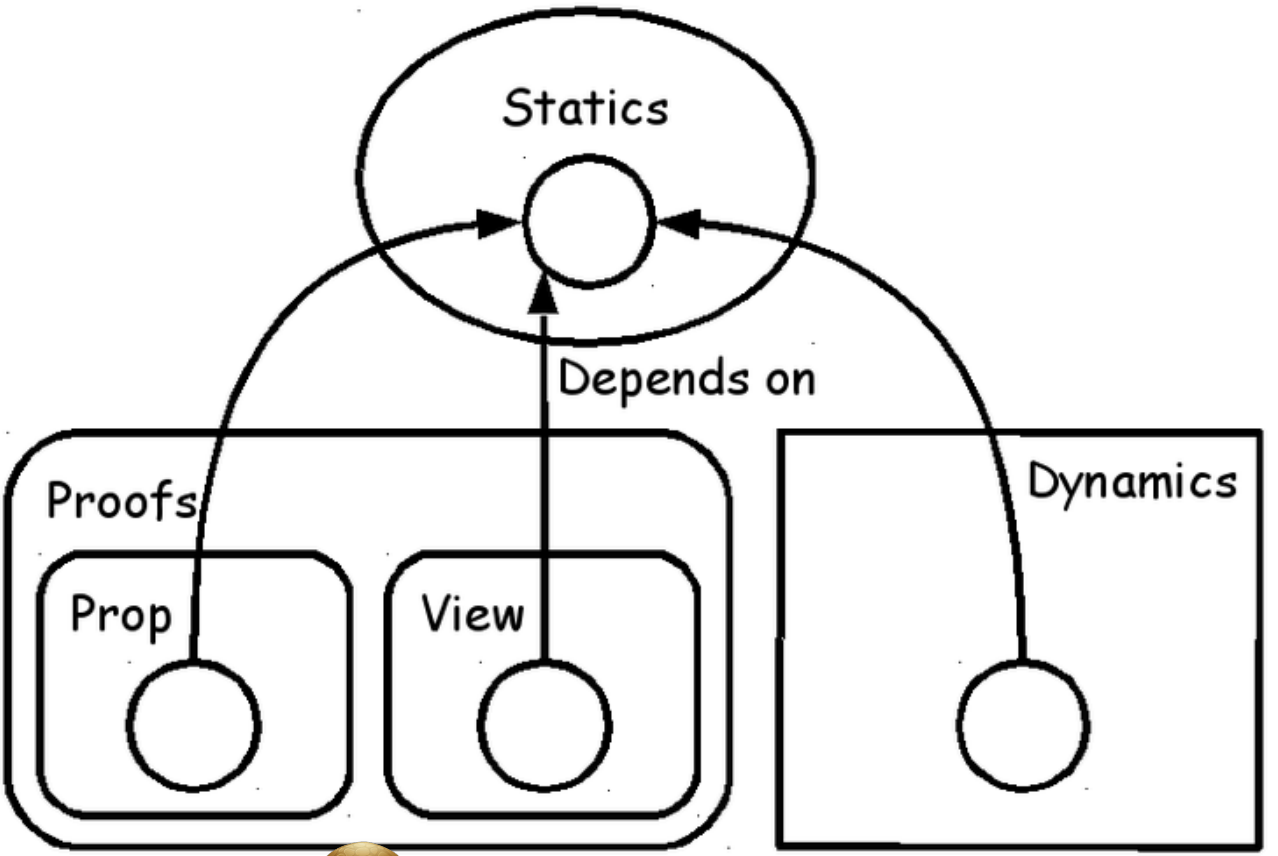
☆ ROM: 32 KB

☆ RAM: 2 KB

What error can be captured by ATS?

```
$ vi ../../SATS/lcd.sats
 11 fun lcd_print {n:int}{i:nat | i < n}{j:nat | i + j <= n}
 12 (lcd: !lcd_t, str: string (n), start: size_t (i), len: size_t (j)): void
$ vi DATS/main.dats
 14   fun loop {n:int}{i:nat | i < n} .<n-i>.
 15            (lcd: !lcd_t, str: string (n), pos: size_t (i)): void = {
 16 //  val () = if pos + i2sz LCD_WIDTH <= length str then {
 17     val () = if pos + i2sz LCD_WIDTH <= 1 + length str then { // Error!
 --snip--
 19       val () = (lcd_setCursor (lcd, 0, 1);
 20                 lcd_print (lcd, str, pos, i2sz LCD_WIDTH))
$ patsopt -o DATS/main_dats.c.tmp -d DATS/main.dats
/home/kiwamu/src/arduino-ats/demo/lcd_greeting/DATS/main.dats: 1016(line=20,
offs=26) -- 1016(line=20, offs=26): error(3): unsolved constraint: C3NSTRprop
(C3TKmain(); S2Eapp(S2Ecst(<=); S2Eapp(S2Ecst(+); S2EVar(1969->S2Evar(i
(5685))), S2EVar(19[illegible]S2Eintinf(16))), S2EVar(1968->S2Evar(n(5684)))))
```

[2] ATS language basics



Dynamics: ML-style programming

☆ Dynamics of ATS is similar to Standard ML.

☆ You should represent type signature of function, because ATS can't inference everything.

☆ You should introduce main function, because ATS code is compiled into C language.

Fizzbuzz on Standard ML

```
local
  fun fbstr i =
    case (i mod 3 = 0, i mod 5 = 0) of
        (true , true ) => "FizzBuzz"
      | (true , false) => "Fizz"
      | (false, true ) => "Buzz"
      | (false, false) => Int.toString i

  fun fizzbuzz' (n, j) =
    if n = j then ()
      else (print (fbstr j ^ "\n"); fizzbuzz' (n, j+1))
in
  fun fizzbuzz n = fizzbuzz' (n, 1)
  val _ = fizzbuzz 100
end
```

Fizzbuzz on ATS

```
local
  fun fbstr (i:int): string =
    case (i mod 3 = 0, i mod 5 = 0) of
        (true , true ) => "FizzBuzz"
      | (true , false) => "Fizz"
      | (false, true ) => "Buzz"
      | (false, false) => tostring_int i

  fun fizzbuzz' (n:int, j:int): void =
    if n = j then ()
      else (println! (fbstr j); fizzbuzz' (n, j+1))
in
  fun fizzbuzz (n:int): void = fizzbuzz' (n, 1)
  val _ = fizzbuzz 100
end
```


ATS binary is portable

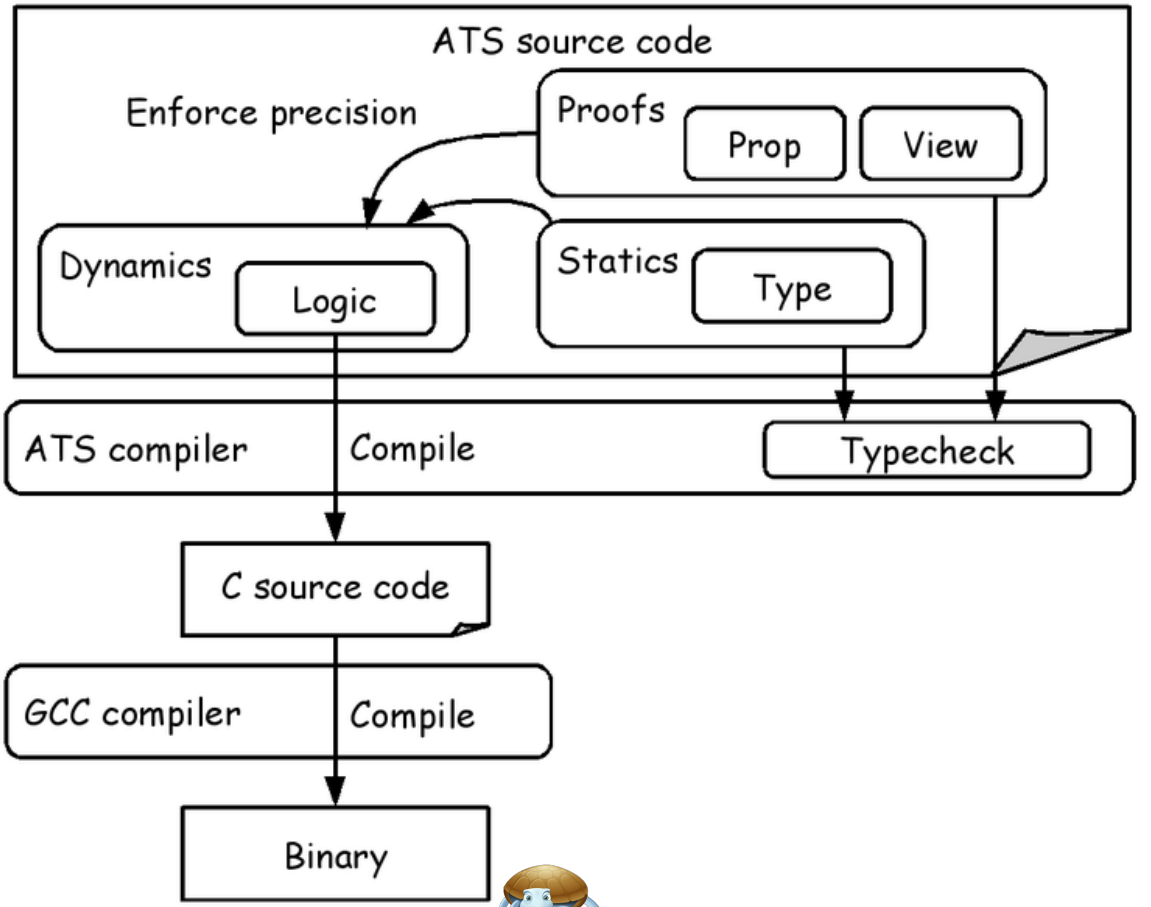
☆ File size is 13 kB:

```
$ ls -lh fizzbuzz
-rwxr-xr-x 1 kiwamu kiwamu 13K Nov  6 21:45 fizzbuzz*
```

☆ Number of undefined symbols is 6:

```
$ nm fizzbuzz | grep 'U '
                 U __libc_start_main@@GLIBC_2.2.5
                 U exit@@GLIBC_2.2.5
                 U fflush@@GLIBC_2.2.5
                 U fprintf@@GLIBC_2.2.5
                 U fwrite@@GLIBC_2.2.5
                 U longjmp@@GLIBC_2.2.5
```


ATS compile flow



What can ATS do?

☆ Write code on bare metal hardware.

☆ Write code in Linux kernel.

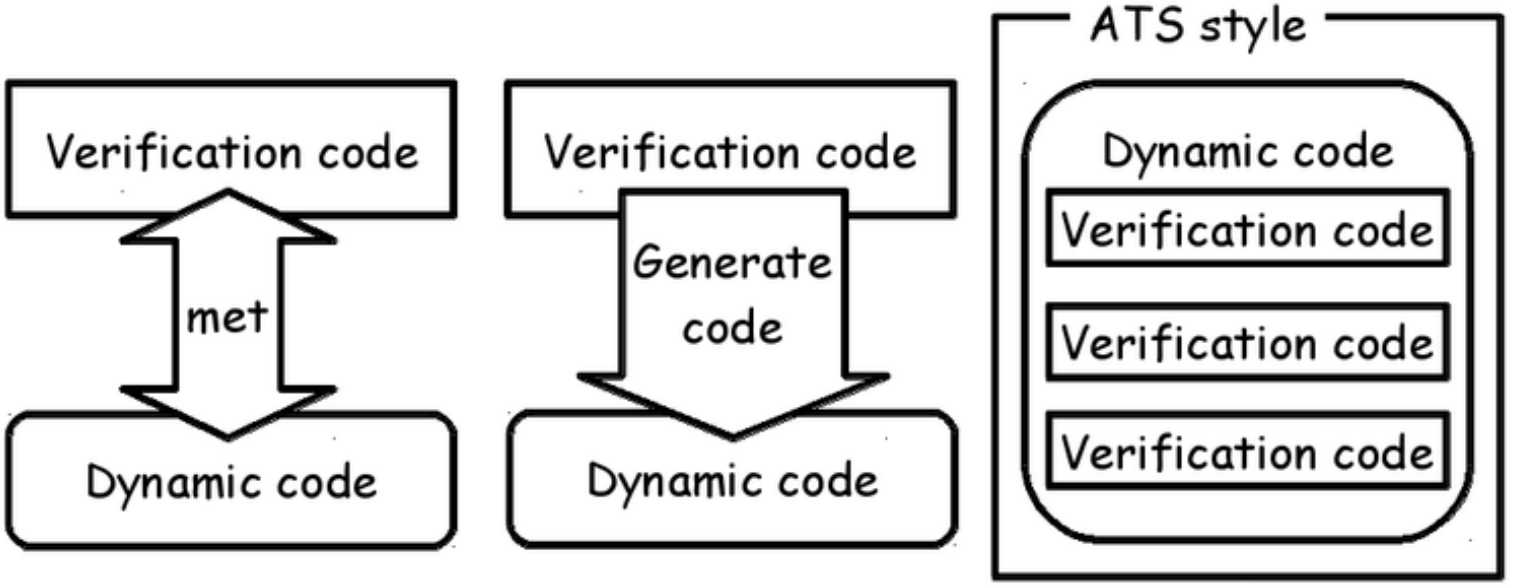
☆ Use strong type without any OS.

☆ Prove code using dependent types.

☆ Safely use malloc using linear types.

☆ Safely use pointer using linear types.

[3] Prop (proof) on ATS



Prop: Curry–Howard in ATS

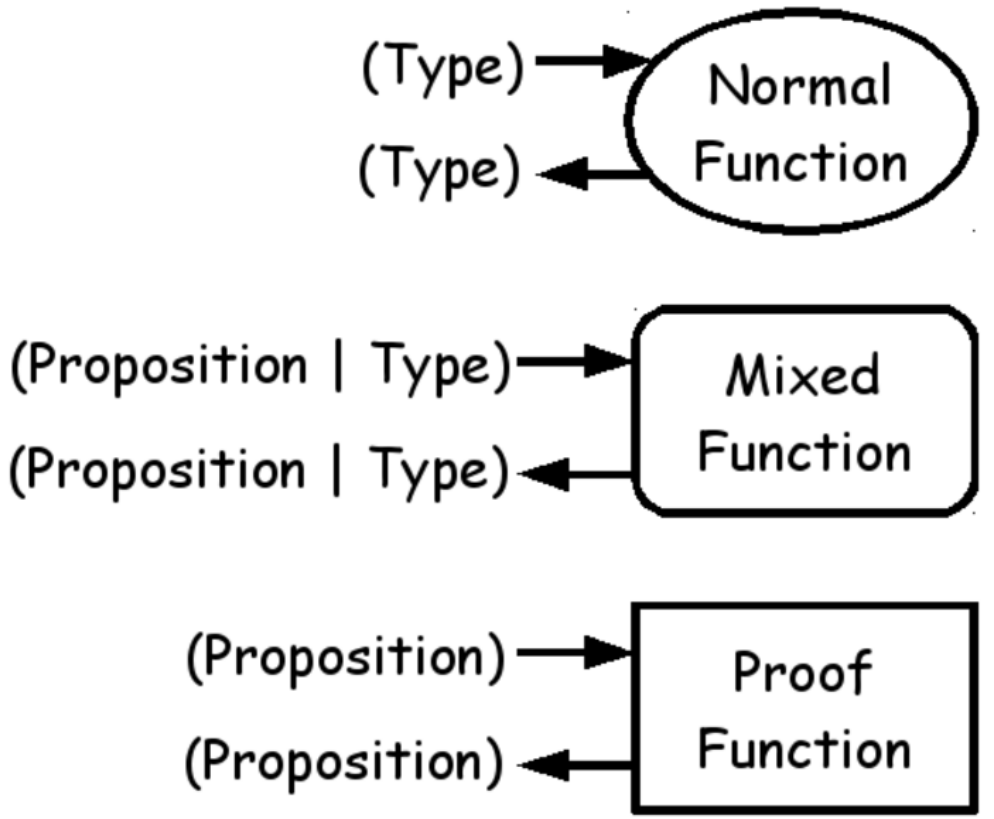
☆ Type: Function signature introduced by keyword "fun"

☆ Program: Function body introduced by keyword "implement"

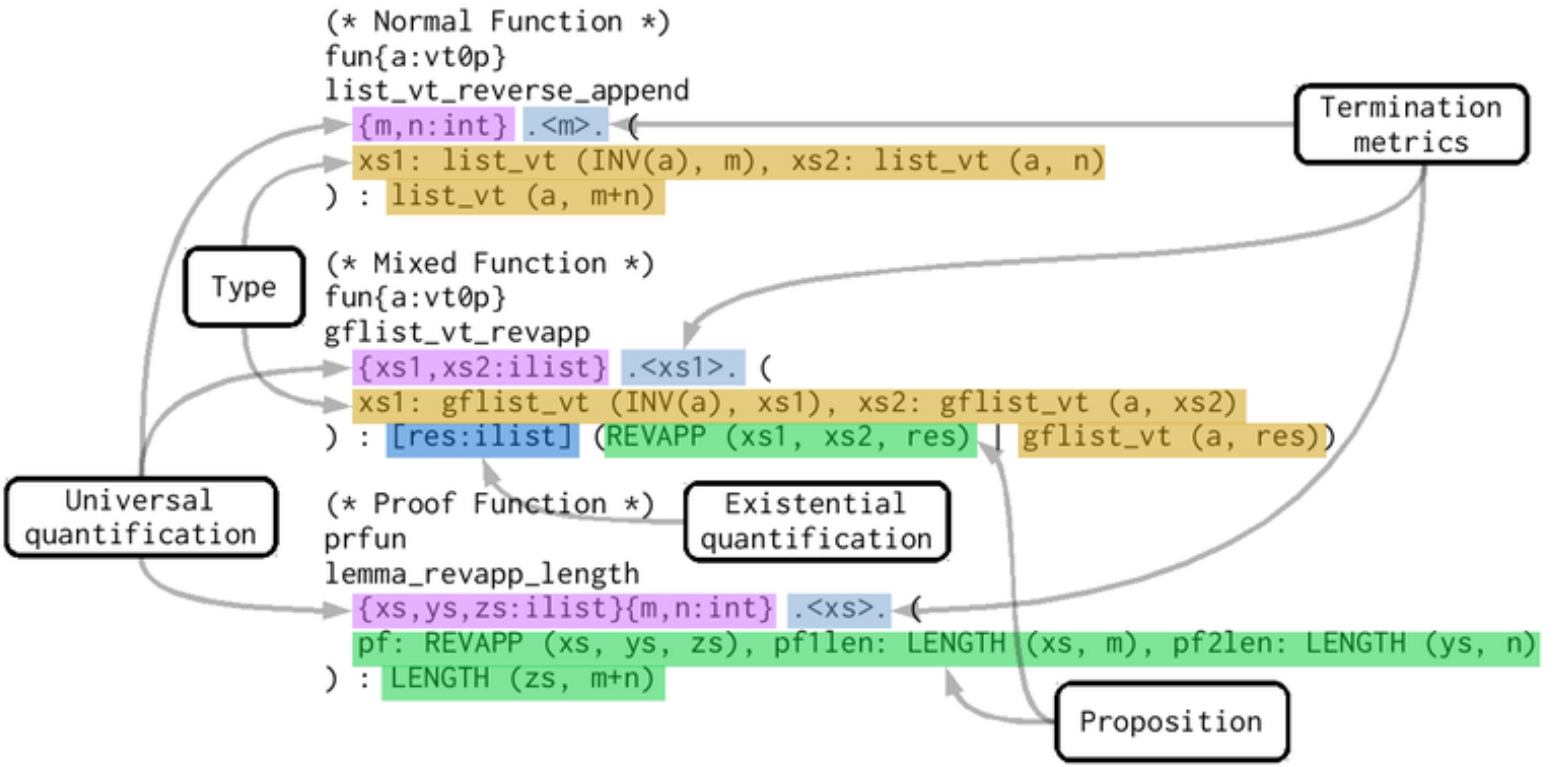
☆ Proposition: Proof function signature introduced by keyword "prfun"

☆ Proof: Proof function body introduced by keyword "primplement"

Prop: Style of functions



Prop: Function signature



Prop: Function body

```
fun{a:vt0p}
split
  {xs:ilist}{n,i:nat | i <= n} .<i>. (
  pflen: LENGTH (xs, n)
| xs: &gflist_vt (a, xs) >> gflist_vt (a, xs1), i: int i
) : #[xs1,xs2:ilist] (
  APPEND (xs1, xs2, xs), LENGTH (xs1, i) | gflist_vt (a, xs2)
) =
  if i > 0 then let
    prval LENGTHcons (pflen) = pflen
    val @gflist_vt_cons (_, xs1) = xs
    val (pfapp, pf1len | xs2) = split (pflen | xs1, i-1)
    prval () = fold@ (xs)
  in
    (APPENDcons (pfapp), LENGTHcons (pf1len) | xs2)
  end else let
    val xs2 = xs
    val () = xs := gflist_vt_nil ()
    prval pfapp = append_unit_left ()
  in
    (pfapp, LENGTHnil () | xs2)
  end
```

Proof

Program

Prop is similar to Coq

☆ Coq is an interactive theorem prover.

☆ https://coq.inria.fr/

☆ ATS also has subsystem for theorem-proving.

☆ But ATS doesn't have tactics.

☆ ATS constructs proofs as total functions.

Prop: Coq code

```
Inductive day : Type :=
  | monday : day
  | tuesday : day
  | wednesday : day
  | thursday : day
  | friday : day
  | saturday : day
  | sunday : day.

Inductive good_day : day -> Prop :=
  | gd_sat : good_day saturday
  | gd_sun : good_day sunday.

Theorem gds : good_day sunday.
Proof. apply gd_sun. Qed.
```


Prop: ATS code

```
datasort Day =
  | Monday
  | Tuesday
  | Wednesday
  | Thursday
  | Friday
  | Saturday
  | Sunday

dataprop Good_Day (Day) =
  | Gd_Sat (Saturday) of ()
  | Gd_Sun (Sunday) of ()

extern prfun gds: Good_Day Sunday
primplement gds = Gd_Sun ()
```


[4] View (linear type) on ATS

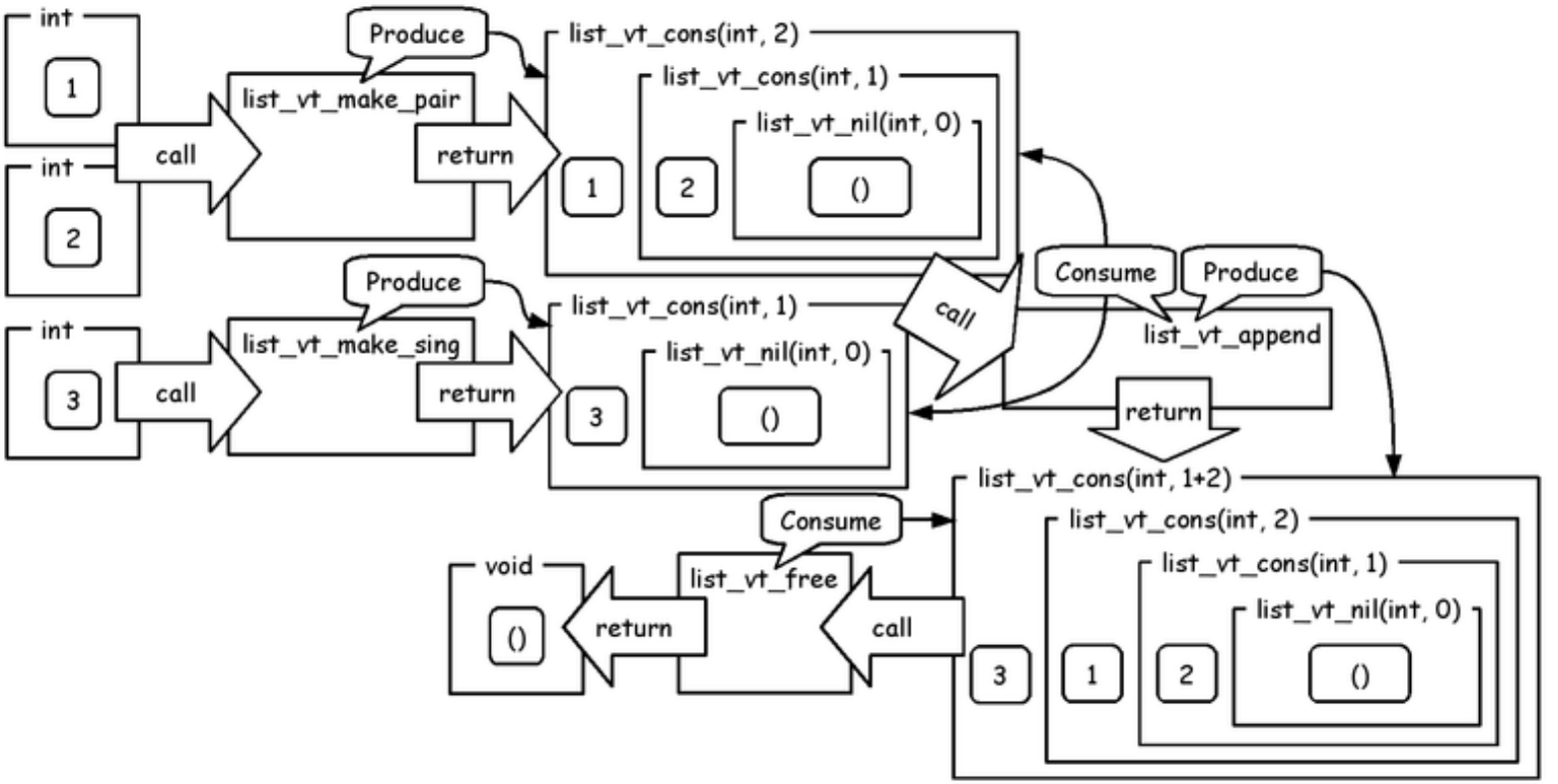
☆ View is linear type in ATS

☆ View manages producing and consuming resource

☆ Example of resource: memory chunk, array, list, queue, lock/unlock, session, ...

☆ At-view is a ticket to permit dereferencing pointer

At-view: Figure of append list



At-view: Code of append list

```
#include "share/atspre_staload.hats"

implement main0 () = {
  val l1 = list_vt_make_pair (1, 2)
  val l2 = list_vt_make_sing 3
  val l3 = list_vt_append (l2, l1)
  val () = print_list_vt<int> l3 (* => 3, 1, 2 *)
(*val () = list_vt_free<int> l3 *) (* Compile-time error! *)
}
```


[5] Compare ATS with Event-B

☆ Event-B generates code from verification.

☆ ATS write verification in code.

☆ Event-B focuses on interaction between contexts.

☆ ATS focuses on specification in single context.

☆ ATS can manage multi-context with some code.

Compare ATS with VeriFast

VeriFast is very similar to ATS, but...

☆ VeriFast is closed source.

☆ ATS is open source.

☆ VeriFast is for C/Java.

☆ ATS is for C/Erlang/JavaScript/PHP/Perl/Python.

☆ VeriFast's host language is C or Java.

☆ ATS's host language is ML.

Compare ATS with XMOS

☆ XMOS strongly focuses on channel implemented on hardware.

☆ ATS also uses strongly typed channel what needs some support by OS.

`http://ats-lang.sourceforge.net/EXAMPLE/EFFECTIVATS/ssntyped-channels-1/main.html`

☆ ATS's channel may be easily used without OS-support, with hardware-support like XMOS.

[6] Conclusion

☆ ATS can use strong type without any OS.

☆ ATS can prove code using dependent types.

☆ ATS can safely use pointer using linear types.

☆ ATS can manage resource using linear types.

☆ Join "ATS logo Japan ATS User Group"!

☆ http://jats-ug.metasepi.org/