# A (Not So Gentle) Introduction To ATS

Aditya Siram

September 26, 2017

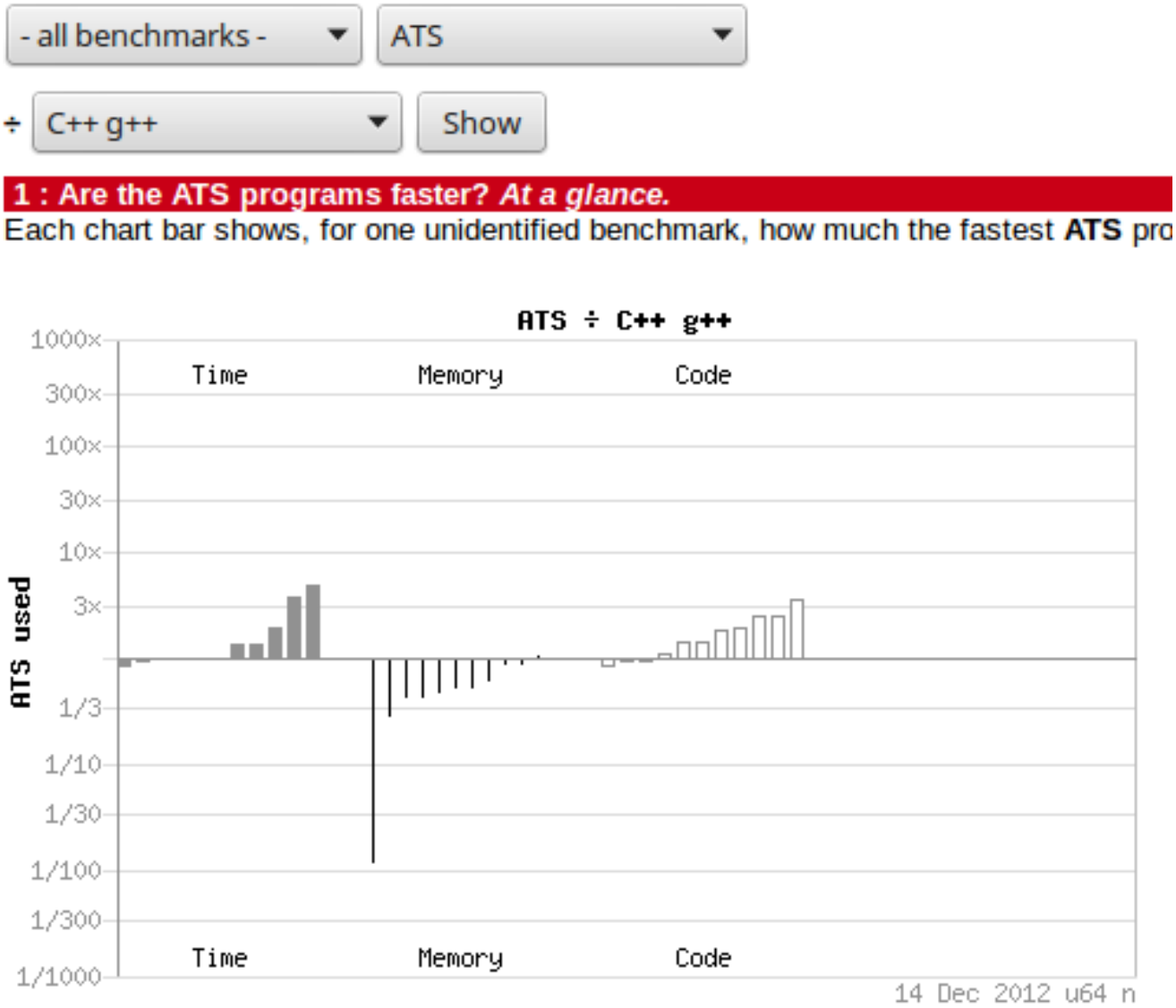
# Outline

# ATS

- The elevator pitch
- No runtime
- Exactly the same performance predictability
  - Decompiles to C
  - No optimizations (except TCO)
  - GCC does the rest
- Exactly the same control of C
  - Pointer arithmetic
  - malloc/free
  - stack allocation
- Completely verified at compile time
  - type system has zero overhead

# ATS

- Alioth benchmark screenshot (unfortunately taken down)



# ATS

- Is an ML (not standard)
  - ADTS, pattern-matching, modules etc.
  - Same power of abstraction
- Linear logic to manage resources
  - Prove it exists, consume proof, repeat
  - file handles, sockets, anything
- But especially memory
  - Prove pointer is initialized, dereference, repeat
  - Type checked pointer arithmetic

# ATS

- Refinement types

```
fun foo
    {                              }
    (i : int   ) ...
```

- Theorem prover
  - Inductive, only.
- Optional GC
  - Won't get in this much in this talk.

# ATS

- Refinement types

```
fun foo
    {                              }
    (i : int n) ...
```

- Theorem prover
  - Inductive, only.
- Optional GC
  - Won't get in this much in this talk.

# ATS

- Refinement types

```
fun foo
    {n:int                          }
    (i : int n) ...
```

- Theorem prover
  - Inductive, only.
- Optional GC
  - Won't get in this much in this talk.

# ATS

- Refinement types

```
fun foo
    {n:int | n > 0          }
    (i : int n) ...
```

- Theorem prover
  - Inductive, only.
- Optional GC
  - Won't get in this much in this talk.

# ATS

- Refinement types

```
fun foo
    {n:int | n > 0 && n < 10}
    (i : int n) ...
```

- Theorem prover
  - Inductive, only.
- Optional GC
  - Won't get in this much in this talk.

# ATS

- Refinement types

```
fun foo
    {n:int | n > 0 && n < 10}
    (i : int n) ...
```

- Theorem prover
  - Inductive, only.
- Optional GC
  - Won't get in this much in this talk.

# ATS

- An ML (not standard)
- Linear logic
- Refinement types
- Theorem proving
- Exactly the same control & performance profile of C

# ATS

- Linear logic to manage resources
  - Prove it exists, consume proof, repeat
  - file handles, sockets, anything
- But especially memory
  - Prove pointer is initialized, dereference, repeat
  - Type checked pointer arithmetic

# ATS

# Hello world

- Minimal

```
implement main0 () = println! "hello world!"
```

# Swap

- A slightly non-standard swap

```
void swap(void *i, void *j, size_t size) {
  void* tmp = malloc(size);
  memcpy(tmp, j, size);
  memcpy(j, i, size);
  memcpy(i, tmp, size);
  free(tmp);
}
```

# Swap

- A slightly non-standard swap

```
void swap(void *i, void *j, size_t size) {
  void* tmp = malloc(size);




}
```

# Swap

- A slightly non-standard swap

```
void swap(void *i, void *j, size_t size) {
  void* tmp = malloc(size);
  memcpy(tmp, j, size);
  memcpy(j, i, size);
  memcpy(i, tmp, size);

}
```

# Swap

- A slightly non-standard swap

```
void swap(void *i, void *j, size_t size) {
  void* tmp = malloc(size);
  memcpy(tmp, j, size);
  memcpy(j, i, size);
  memcpy(i, tmp, size);
  free(tmp);
}
```

# Swap

- A slightly non-standard swap

```
%{
  #include <stdio.h>
  void swap(void *i, void *j, size_t size) {
    ...
  }
%}
```

# Swap

- A slightly non-standard swap

```
%{
  #include <stdio.h>
  void swap(void *i, void *j, size_t size) {
    ...
  }
%}
extern fun swap (i:ptr, j:ptr, s:size_t) : void = "ext#swap"
```

# Swap

- A slightly non-standard swap

```
%{
  #include <stdio.h>
  void swap(void *i, void *j, size_t size) {
    ...
  }
%}
extern fun swap (i:ptr, j:ptr, s:size_t) : void = "ext#swap"
extern fun malloc(s:size_t):ptr = "ext#malloc"
```

# Swap

- Runner

```
implement main0 () =
  let
     val i = malloc(sizeof<int>)
     val j = malloc(sizeof<double>)
     val _ = swap(i,j,sizeof<double>)
  in
     ()
  end
```

# Swap

- Runner

```
implement main0 () =
  let
     val i = malloc(sizeof<int>) // all good


  in

  end
```

# Swap

- Runner

```
implement main0 () =
  let
     val i = malloc(sizeof<int>)
     val j = malloc(sizeof<double>) // uh oh!

  in

  end
```

# Swap

- Runner

```
implement main0 () =
  let
     val i = malloc(sizeof<int>)
     val j = malloc(sizeof<double>)
     val _ = swap(i,j,sizeof<double>) // oh noes!
  in

  end
```

# Swap

- Runner

```
implement main0 () =
  let
     val i = malloc(sizeof<int>)
     val j = malloc(sizeof<double>)
     val _ = swap(i,j,sizeof<double>)
  in
     () // free as in leak
  end
```

# Swap

- Safe swap

```
extern fun swap (i:ptr, j:ptr, s:size_t) : void = "ext#swap"
```

# Swap

- Safe swap

```
extern fun swap                                    : void = "ext#swap"
```

# Swap

- Safe swap

```
extern fun swap                                    :          = "ext#swap"
```

# Swap

- Safe swap

```
extern fun swap
```

```
= "ext#swap"
```

# Swap

- Safe swap

```
extern fun swap
  {a : t@ype}
```

```
= "ext#swap"
```

# Swap

- Safe swap

```
extern fun swap
  {a : t@ype}
  {l1: addr |                }
                                = "ext#swap"
```

# Swap

- Safe swap

```
extern fun swap
  {a : t@ype}
  {l1: addr | l1 > null}
```

```
= "ext#swap"
```

# Swap

- Safe swap

```
extern fun swap
  {a : t@ype}
  {l1: addr | l1 > null}
  {l2: addr | l2 > null}
                                  = "ext#swap"
```

# Swap

- Safe swap

```
extern fun swap
  {a : t@ype}
  {l1: addr | l1 > null}
  {l2: addr | l2 > null}
  (                  i : ptr l1                              ):
                              = "ext#swap"
```

# Swap

- Safe swap

```
extern fun swap
  {a : t@ype}
  {l1: addr | l1 > null}
  {l2: addr | l2 > null}
  (                    i : ptr l1, j : ptr l2                    ):
                              = "ext#swap"
```

# Swap

- Safe swap

```
extern fun swap
  {a : t@ype}
  {l1: addr | l1 > null}
  {l2: addr | l2 > null}
  (                    i : ptr l1, j : ptr l2, s: sizeof_t a):
                                  = "ext#swap"
```

# Swap

- Safe swap

```
extern fun swap
  {a : t@ype}
  {l1: addr | l1 > null}
  {l2: addr | l2 > null}
  (                  | i : ptr l1, j : ptr l2, s: sizeof_t a):
                                  = "ext#swap"
```

# Swap

- Safe swap

```
extern fun swap
  {a : t@ype}
  {l1: addr | l1 > null}
  {l2: addr | l2 > null}
  (a @ l1           | i : ptr l1, j : ptr l2, s: sizeof_t a):
                              = "ext#swap"
```

# Swap

- Safe swap

```
extern fun swap
  {a : t@ype}
  {l1: addr | l1 > null}
  {l2: addr | l2 > null}
  (a @ l1 , a @ l2 | i : ptr l1, j : ptr l2, s: sizeof_t a):
                                  = "ext#swap"
```

# Swap

- Safe swap

```
extern fun swap
  {a : t@ype}
  {l1: addr | l1 > null}
  {l2: addr | l2 > null}
  (a @ l1 , a @ l2 | i : ptr l1, j : ptr l2, s: sizeof_t a):
    (                          ) = "ext#swap"
```

# Swap

- Safe swap

```
extern fun swap
  {a : t@ype}
  {l1: addr | l1 > null}
  {l2: addr | l2 > null}
  (a @ l1 , a @ l2 | i : ptr l1, j : ptr l2, s: sizeof_t a):
    (                    void) = "ext#swap"
```

# Swap

- Safe swap

```
extern fun swap
  {a : t@ype}
  {l1: addr | l1 > null}
  {l2: addr | l2 > null}
  (a @ l1 , a @ l2 | i : ptr l1, j : ptr l2, s: sizeof_t a):
    (                | void) = "ext#swap"
```

# Swap

- Safe swap

```
extern fun swap
  {a : t@ype}
  {l1: addr | l1 > null}
  {l2: addr | l2 > null}
  (a @ l1 , a @ l2 | i : ptr l1, j : ptr l2, s: sizeof_t a):
    (a @ l1          | void) = "ext#swap"
```

# Swap

- Safe swap

```
extern fun swap
  {a : t@ype}
  {l1: addr | l1 > null}
  {l2: addr | l2 > null}
  (a @ l1 , a @ l2 | i : ptr l1, j : ptr l2, s: sizeof_t a):
    (a @ l1, a @ l2 | void) = "ext#swap"
```

# Swap

- Safe swap

```
extern fun malloc(s:size_t):ptr = "ext#malloc"
```

# Swap

- Safe swap

```
extern fun malloc
```

```
= "ext#malloc"
```

# Swap

- Safe swap

```
extern fun malloc
       {a:t@ype}


   = "ext#malloc"
```

# Swap

- Safe swap

```
extern fun malloc
       {a:t@ype}
       (s:sizeof_t a):

   = "ext#malloc"
```

# Swap

- Safe swap

```
extern fun malloc
       {a:t@ype}
       (s:sizeof_t a):
                              (          ptr l)
   = "ext#malloc"
```

# Swap

- Safe swap

```
extern fun malloc
       {a:t@ype}
       (s:sizeof_t a):
                              (a? @ l | ptr l)
  = "ext#malloc"
```

# Swap

- Safe swap

```
extern fun malloc
       {a:t@ype}
       (s:sizeof_t a):
       [                    ] (a? @ l | ptr l)
   = "ext#malloc"
```

# Swap

- Safe swap

```
extern fun malloc
       {a:t@ype}
       (s:sizeof_t a):
       [l:addr                ] (a? @ l | ptr l)
   = "ext#malloc"
```

# Swap

- Safe swap

```
extern fun malloc
       {a:t@ype}
       (s:sizeof_t a):
       [l:addr | l > null] (a? @ l | ptr l)
   = "ext#malloc"
```

# Swap

- Safe swap

```
implement main0 () = let
  val (      i) = malloc (sizeof<int>)




in



end
```

# Swap

- Safe swap

```
implement main0 () = let
  val (     | i) = malloc (sizeof<int>)




in



end
```

# Swap

- Safe swap

```
implement main0 () = let
  val (pfi | i) = malloc (sizeof<int>)




in



end
```

# Swap

- Safe swap

```
implement main0 () = let
  val (pfi | i) = malloc (sizeof<int>)
  val (pfj | j) = malloc (sizeof<int>)



in


end
```

# Swap

- Safe swap

```
implement main0 () = let
  val (pfi | i) = malloc (sizeof<int>)
  val (pfj | j) = malloc (sizeof<int>)
  val             = ptr_set(      i, 1)


in


end
```

# Swap

- Safe swap

```
implement main0 () = let
  val (pfi | i) = malloc (sizeof<int>)
  val (pfj | j) = malloc (sizeof<int>)
  val           = ptr_set(pfi | i, 1)


in


end
```

# Swap

- Safe swap

```
implement main0 () = let
  val (pfi | i) = malloc (sizeof<int>)
  val (pfj | j) = malloc (sizeof<int>)
  val (pfi1 | _) = ptr_set(pfi | i, 1)
  val (pfj1 | _) = ptr_set(pfj | j, 2)

in


end
```

# Swap

- Safe swap

```
implement main0 () = let
  val (pfi | i) = malloc (sizeof<int>)
  val (pfj | j) = malloc (sizeof<int>)
  val (pfi1 | _) = ptr_set(pfi | i, 1)
  val (pfj1 | _) = ptr_set(pfj | j, 2)
  val                    = swap(                 i, j, sizeof<int>)
in


end
```

# Swap

- Safe swap

```
implement main0 () = let
  val (pfi | i) = malloc (sizeof<int>)
  val (pfj | j) = malloc (sizeof<int>)
  val (pfi1 | _) = ptr_set(pfi | i, 1)
  val (pfj1 | _) = ptr_set(pfj | j, 2)
  val                    = swap(              | i, j, sizeof<int>)
in


end
```

# Swap

- Safe swap

```
implement main0 () = let
  val (pfi | i) = malloc (sizeof<int>)
  val (pfj | j) = malloc (sizeof<int>)
  val (pfi1 | _) = ptr_set(pfi | i, 1)
  val (pfj1 | _) = ptr_set(pfj | j, 2)
  val                  = swap(pfi1        | i, j, sizeof<int>)
in


end
```

# Swap

- Safe swap

```
implement main0 () = let
  val (pfi | i) = malloc (sizeof<int>)
  val (pfj | j) = malloc (sizeof<int>)
  val (pfi1 | _) = ptr_set(pfi | i, 1)
  val (pfj1 | _) = ptr_set(pfj | j, 2)
  val                    = swap(pfi1, pfj2 | i, j, sizeof<int>)
in


end
```

# Swap

- Safe swap

```
implement main0 () = let
  val (pfi | i) = malloc (sizeof<int>)
  val (pfj | j) = malloc (sizeof<int>)
  val (pfi1 | _) = ptr_set(pfi | i, 1)
  val (pfj1 | _) = ptr_set(pfj | j, 2)
  val (              ()) = swap(pfi1, pfj2 | i, j, sizeof<int>)
in


end
```

# Swap

- Safe swap

```
implement main0 () = let
  val (pfi | i) = malloc (sizeof<int>)
  val (pfj | j) = malloc (sizeof<int>)
  val (pfi1 | _) = ptr_set(pfi | i, 1)
  val (pfj1 | _) = ptr_set(pfj | j, 2)
  val (pfi2      | _) = swap(pfi1, pfj1 | i, j, sizeof<int>)
in


end
```

# Swap

- Safe swap

```
implement main0 () = let
  val (pfi | i) = malloc (sizeof<int>)
  val (pfj | j) = malloc (sizeof<int>)
  val (pfi1 | _) = ptr_set(pfi | i, 1)
  val (pfj1 | _) = ptr_set(pfj | j, 2)
  val (pfi2,pfj2 | _) = swap(pfi1, pfj1 | i, j, sizeof<int>)
in


end
```

# Swap

- Safe swap

```
implement main0 () = let
  val (pfi | i) = malloc (sizeof<int>)
  val (pfj | j) = malloc (sizeof<int>)
  val (pfi1 | _) = ptr_set(pfi | i, 1)
  val (pfj1 | _) = ptr_set(pfj | j, 2)
  val (pfi2,pfj2| _) = swap(pfi1, pfj2 | i, j, sizeof<int>)
in
  free(pfi2 | i);

end
```

# Swap

- Safe swap

```
implement main0 () = let
  val (pfi | i) = malloc (sizeof<int>)
  val (pfj | j) = malloc (sizeof<int>)
  val (pfi1 | _) = ptr_set(pfi | i, 1)
  val (pfj1 | _) = ptr_set(pfj | j, 2)
  val (pfi2,pfj2| _) = swap(pfi1, pfj1 | i, j, sizeof<int>)
in
  free(pfi2 | i);
  free(pfj2 | j);
end
```

# Swap

- Safe swap

```
implement main0 () = let
  val (pfi    ) = malloc
       ^^^
  val (pfi1 | _) = ptr_set(pfi |     )
       ^^^^                ^^^

in


end
```

# Swap

- Safe swap

```
implement main0 () = let


  val (pfi1 | _) =
       ^^^^
  val (pfi2,     | _) == swap(pfi1,       |                    )
in     ^^^^                    ^^^^


end
```

# Swap

- Safe swap

```
implement main0 () = let




  val (pfi2,     | _) =
in      ^^^^
  free(pfi2 |  );
       ^^^^
end
```

# Factorial

- Factorial

```
fun factorial
    { n : int | n >= 1 }
    (i : int n) : double =
  let
    fun loop
        { n : int | n >= 1 }
        .<n>.
        (acc : double, i : int (n)) : double =
      case- i of
      | 1 => acc
      | i when i > 1 => loop(acc * i, i - 1)

  in
    loop(1.0, i)
  end
```

# Factorial

- Factorial

```
fun factorial


  let
    fun loop









  in
    loop(1.0, i)
  end
```

# Factorial

- Factorial

```
fun factorial
    { n : int | n >= 1 }

  let
    fun loop
```

```
  in
    loop(1.0, i)
  end
```

# Factorial

- Factorial

```
fun factorial
    { n : int | n >= 1 }
    (i : int n) : double =
  let
    fun loop
  in
    loop(1.0, i)
  end
```

# Factorial

- Factorial

```
fun factorial
    { n : int | n >= 1 }
    (i : int n) : double =
  let
    fun loop
        { n : int | n >= 1 }
```

```
  in
    loop(1.0, i)
  end
```

# Factorial

- Factorial

```
fun factorial
    { n : int | n >= 1 }
    (i : int n) : double =
  let
    fun loop
        { n : int | n >= 1 }

        (acc : double, i : int (n)) : double =




  in
    loop(1.0, i)
  end
```

# Factorial

- Factorial

```
fun factorial
    { n : int | n >= 1 }
    (i : int n) : double =
  let
    fun loop
        { n : int | n >= 1 }
        .<n>.
        (acc : double, i : int (n)) : double =




  in
    loop(1.0, i)
  end
```

# Factorial

- Factorial

```
fun factorial
    { n : int | n >= 1 }
    (i : int n) : double =
  let
    fun loop
        { n : int | n >= 1 }
        .<n>.
        (acc : double, i : int (n)) : double =
      case- i of



  in
    loop(1.0, i)
  end
```

# Factorial

- Factorial

```
fun factorial
    { n : int | n >= 1 }
    (i : int n) : double =
  let
    fun loop
        { n : int | n >= 1 }
        .<n>.
        (acc : double, i : int (n)) : double =
      case- i of
      | 1 => acc
      |

  in
    loop(1.0, i)
  end
```

# Factorial

- Factorial

```
fun factorial
    { n : int | n >= 1 }
    (i : int n) : double =
  let
    fun loop
        { n : int | n >= 1 }
        .<n>.
        (acc : double, i : int (n)) : double =
      case- i of
      | 1 => acc
      | i

  in
    loop(1.0, i)
  end
```

# Factorial

- Factorial

```
fun factorial
    { n : int | n >= 1 }
    (i : int n) : double =
  let
    fun loop
        { n : int | n >= 1 }
        .<n>.
        (acc : double, i : int (n)) : double =
      case- i of
      | 1 => acc
      | i when i > 1

  in
    loop(1.0, i)
  end
```

# Factorial

- Factorial

```
fun factorial
    { n : int | n >= 1 }
    (i : int n) : double =
  let
    fun loop
        { n : int | n >= 1 }
        .<n>.
        (acc : double, i : int (n)) : double =
      case- i of
      | 1 => acc
      | i when i > 1 => loop(acc * i, i - 1)

  in
    loop(1.0, i)
  end
```

# Factorial

- Factorial

```
fun factorial


  let
    fun loop
        { n : int | n >= 1 } <---


      case- i of
      |
      | i when i > 1 => loop(acc * i, i - 1)
          ^^^^^^^^^^
  in
    loop(1.0, i)
  end
```

# Factorial

- Factorial

```
fun factorial


  let
    fun loop
        { n : int | n >= 1 } <---


      case- i of
      |
      | i when i > 1 => loop(acc * i, i - 1)
                                     ^^^^^

  in
    loop(1.0, i)
  end
```

# Factorial

- Factorial

```
fun factorial


  let
    fun loop

        .<n>. <---

      case- i of
      |
      | i when i > 1 => loop(acc * i, i + 1)
                                      ^^^^^

  in
    loop(1.0, i)
  end
```

# Dataviewtype

- ADT describing an array of pointers

```
dataview array_v
(
  a:t@ype,
  l: addr,
  n : int
) =
    array_v_nil  (a, l, 0)
  | array_v_cons (a, l, n) of
      (a @ l, array_v (a, l+sizeof a, n-1))
```

# Dataviewtype

- ADT describing an array of pointers

```
dataview array_v
(



) =
```

# Dataviewtype

- ADT describing an array of pointers

```
dataview array_v
(



) =
    array_v_nil
  |
```

# Dataviewtype

- ADT describing an array of pointers

```
dataview array_v
(



) =
    array_v_nil
  | array_v_cons
```

# Dataviewtype

- ADT describing an array of pointers

```
dataview array_v
(
  a:t@ype,


) =
    array_v_nil
  | array_v_cons
```

# Dataviewtype

- ADT describing an array of pointers

```
dataview array_v
(
  a:t@ype,
  l: addr,

) =
    array_v_nil
  | array_v_cons
```

# Dataviewtype

- ADT describing an array of pointers

```
dataview array_v
(
  a:t@ype,
  l: addr,
  n : int
) =
    array_v_nil
  | array_v_cons
```

# Dataviewtype

- ADT describing an array of pointers

```
dataview array_v
(
  a:t@ype,
  l: addr,
  n : int
) =
    array_v_nil  (a, l, 0)
  | array_v_cons
```

# Dataviewtype

- ADT describing an array of pointers

```
dataview array_v
(
  a:t@ype,
  l: addr,
  n : int
) =
    array_v_nil  (a, l, 0)
  | array_v_cons (a, l, n)
```

# Dataviewtype

- ADT describing an array of pointers

```
dataview array_v
(
  a:t@ype,
  l: addr,
  n : int
) =
    array_v_nil  (a, l, 0)
  | array_v_cons (a, l, n) of
       (a @ l,                              )
```

# Dataviewtype

- ADT describing an array of pointers

```
dataview array_v
(
  a:t@ype,
  l: addr,
  n : int
) =
    array_v_nil  (a, l, 0)
  | array_v_cons (a, l, n) of
       (a @ l, array_v (                    ))
```

# Dataviewtype

- ADT describing an array of pointers

```
dataview array_v
(
  a:t@ype,
  l: addr,
  n : int
) =
    array_v_nil  (a, l, 0)
  | array_v_cons (a, l, n) of
       (a @ l, array_v (a,                  ))
```

# Dataviewtype

- ADT describing an array of pointers

```
dataview array_v
(
  a:t@ype,
  l: addr,
  n : int
) =
    array_v_nil  (a, l, 0)
  | array_v_cons (a, l, n) of
       (a @ l, array_v (a, l+sizeof a,    ))
```

# Dataviewtype

- ADT describing an array of pointers

```
dataview array_v
(
  a:t@ype,
  l: addr,
  n : int
) =
    array_v_nil  (a, l, 0)
  | array_v_cons (a, l, n) of
      (a @ l, array_v (a, l+sizeof a, n-1))
```