A Tase Of ATS

Aditya Siram

June 24, 2019

Outline

ATS

- An ML with ADTs, pattern matching, tail calls
- Can be exactly as good the C equivalent
  - Control over memory
  - Performance
- And type safe.

ATS

- Compiles to *predictable* C
  - Recursion is well supported
- Compiles to *predictable* C
  - Allows C idioms
  - malloc/free, pointers, stack control
- No compiler optimizations except TCO
  - Almost no ...
- Linear/refinement types, proof level language

ATS

- Extremely difficult
  - Syntax
  - Errors
- But I want to get into the more interesting features

ATS

```
implement main0(argc,argv) =
  let
    val a = fopen("test.txt","r")
    val b = fopen("test.txt","r")
    var f = lam@(s:string):void => println! s
  in (
    fwithline(a,f);
    fclose(a);
    fclose(b)
  )
  end
```

ATS

```
implement main0(argc,argv) =
  let
    val a = fopen("test.txt","r")


  in (



  )
  end
```

ATS

```
datavtype FileHandle = FileHandle of ()
```

ATS

```
fun fopen(path:string,mode:string): FileHandle =
  let
    extern castfn toFileHandle(p:ptr0):<> FileHandle
  in
    toFileHandle($extfcall(ptr0,"fopen",path,mode))
  end
```

ATS

```
fun fopen(path:string,mode:string): FileHandle =
  let

  in
                ($extfcall(ptr0,"fopen",path,mode))
  end
```

ATS

```
fun fopen(path:string,mode:string): FileHandle =
  let
                  toFileHandle(p:ptr0):<> FileHandle
  in
    toFileHandle($extfcall(ptr0,"fopen",path,mode))
  end
```

ATS

```
fun fopen(path:string,mode:string): FileHandle =
  let
    extern castfn toFileHandle(p:ptr0):<> FileHandle
  in
    toFileHandle($extfcall(ptr0,"fopen",path,mode))
  end
```

ATS

```
implement main0(argc,argv) =
  let
    val a = fopen("test.txt","r")


  in (



  )
  end
```

ATS

```
implement main0(argc,argv) =
  let
    val a = fopen("test.txt","r")
    val b = fopen("test.txt","r")

  in (



  )
  end
```

ATS

```
implement main0(argc,argv) =
  let
    val a = fopen("test.txt","r")
    val b = fopen("test.txt","r")
    var f = lam@(s:string):void => println! s
  in (          |
                +-------- stack allocated closure!


  )
  end
```

ATS

```
implement main0(argc,argv) =
  let
    val a = fopen("test.txt","r")
    val b = fopen("test.txt","r")
    var f = lam@(s:string):void => println! s
  in (
    fwithline(a,f);


  )
  end
```

ATS

```
fun fwithline(
    fh: !FileHandle,
    f: &(string) -<clo1> void
    ):void =
  let









  in

  end
```

ATS

```
fun fwithline(
    fh: !FileHandle,
    f: &(string) -<clo1> void
    ):void =
  let

    val _ = $extfcall(int,"getline",        ,     ,          )
  in

  end
```

ATS

```
fun fwithline(
    fh: !FileHandle,
    f: &(string) -<clo1> void
    ):void =
  let
    var len = i2sz(0)
    val lenP = addr@len



    val _ = $extfcall(int,"getline",        ,lenP,          )
  in

  end
```

ATS

```
fun fwithline(
    fh: !FileHandle,
    f: &(string) -<clo1> void
    ):void =
  let
    var len = i2sz(0)
    val lenP = addr@len
    var buffer = the_null_ptr
    val bufferP = addr@buffer

    val _ = $extfcall(int,"getline",bufferP,lenP,          )
  in

  end
```

ATS

```
fun fwithline(
    fh: !FileHandle,
    f: &(string) -<clo1> void
    ):void =
  let
    var len = i2sz(0)
    val lenP = addr@len
    var buffer = the_null_ptr
    val bufferP = addr@buffer
                  toPtr{l:addr}(f: !FileHandle):<> ptr0
    val _ = $extfcall(int,"getline",bufferP,lenP,toPtr(fh))
  in

  end
```

ATS

```
fun fwithline(
    fh: !FileHandle,
    f: &(string) -<clo1> void
    ):void =
  let
    var len = i2sz(0)
    val lenP = addr@len
    var buffer = the_null_ptr
    val bufferP = addr@buffer
    extern castfn toPtr{l:addr}(f: !FileHandle):<> ptr0
    val _ = $extfcall(int,"getline",bufferP,lenP,toPtr(fh))
  in

  end
```

ATS

```
fun fwithline(
    fh: !FileHandle,
    f: &(string) -<clo1> void
    ):void =
  let


    var buffer = the_null_ptr



  in
    f (                          (buffer))
  end
```

ATS

```
fun fwithline(
    fh: !FileHandle,
    f: &(string) -<clo1> void
    ):void =
  let


    var buffer = the_null_ptr



  in
    f ($UN.castvwtp0{string}(buffer))
  end
```

ATS

```
implement main0(argc,argv) =
  let
    val a = fopen("test.txt","r")
    val b = fopen("test.txt","r")
    var f = lam@(s:string):void => println! s
  in (
    fwithline(a,f);


  )
  end
```

ATS

```
implement main0(argc,argv) =
  let
    val a = fopen("test.txt","r")
    val b = fopen("test.txt","r")
    var f = lam@(s:string):void => println! s
  in (
    fwithline(a,f);
    fclose(a);

  )
  end
```

ATS

```
fun fclose(f:FileHandle):void =
  let
    extern castfn fromFH(f:FileHandle):<> ptr0
  in
    $extfcall(void,"fclose",fromFH(f))
  end
```

ATS

```
implement main0(argc,argv) =
  let
    val a = fopen("test.txt","r")
    val b = fopen("test.txt","r")
    var f = lam@(s:string):void => println! s
  in (
    fwithline(a,f);
    fclose(a);
    fclose(b)
  )
  end
```

```
fun fwithline(
    fh: !FileHandle,

    ):void =

fun fclose(f: FileHandle):void =
```

ATS

```
datavtype arr(a:vtflt,addr,int) =
 | {l:addr}
   arr_nil(a,l,0) of ()
 | {l:addr}{n:nat}
   arr_cons(a,l,n+1) of (a,arr(a,l+sizeof(a),n))
```

ATS

```
datavtype arr(a:vtflt,addr,int) =
 |
   arr_nil(      ) of ()
 |
   arr_cons(        ) of (                          )
```

ATS

```
datavtype arr(a:vtflt,addr,int) =
 | {l:addr}
   arr_nil(a,l,0) of ()
 |
   arr_cons(       ) of (                    )
```

ATS

```
datavtype arr(a:vtflt,addr,int) =
 | {l:addr}
   arr_nil(a,l,0) of ()
 | {l:addr}{n:nat}
   arr_cons(       ) of (                    )
```

ATS

```
datavtype arr(a:vtflt,addr,int) =
 | {l:addr}
   arr_nil(a,l,0) of ()
 | {l:addr}{n:nat}
   arr_cons(a,l,n+1) of (a,arr(                    ))
```

ATS

```
datavtype arr(a:vtflt,addr,int) =
 | {l:addr}
   arr_nil(a,l,0) of ()
 | {l:addr}{n:nat}
   arr_cons(a,l,n+1) of (a,arr(a,                 ))
```

ATS

```
datavtype arr(a:vtflt,addr,int) =
 | {l:addr}
   arr_nil(a,l,0) of ()
 | {l:addr}{n:nat}
   arr_cons(a,l,n+1) of (a,arr(a,l+sizeof(a)  ))
```

ATS

```
datavtype arr(a:vtflt,addr,int) =
 | {l:addr}
   arr_nil(a,l,0) of ()
 | {l:addr}{n:nat}
   arr_cons(a,l,n+1) of (a,arr(a,l+sizeof(a),n))
```

ATS

```
fun {a:tflt} arr_init
  {n:nat}(n:size(n), init:a): [a:vtflt][l:addr] arr(a,l,n) =
  let
    val p0 = $extfcall(cptr(a),"calloc",n*sizeof<a>,sizeof<a>)
    fun loop(p0:cptr(a),p_end:cptr(a),init:a):void =
      if (p0 < p_end) then (
        $UN.cptr0_set<a>(p0,init);
        loop(succ(p0), p_end, init)
      )
      else ()
    val () = loop(p0,p0+n*sizeof<a>,init)
  in
    $UN.castvwtp0(p0)
  end
```