Who's Got Your Mail? Mr. Mime!

Romain Calascibetta

Abstract

Even in these modern times, email remains an incredibly popular messaging medium. Estimates range wildly but it is common to see over 100 billion emails sent per day reported. It's mature and stable design has resulted in its growth over time to the point where perhaps half the planet uses email in some form. Email standards have themselves also evolved over 35 years to support increasingly wide range of usages. From RFC822 [1] to its many modern day extensions, email has become surprisingly complex.

MrMime is a new from-scratch pure-OCaml library that targets parsing any and all emails. Driven by the complexity of parsing modern emails, and multi-part mail in particular, it is built using Continuation Passing Style (CPS) throughout, assembling small parser combinators on-the-fly to build a parser for the mail based on the contents of the mail headers. With this we can create MirageOS unikernels that send, receive, store and process emails, paving the way for such unikernel deployments as SMTP proxies, spam filters and more.

MrMime has been used to parse over 1 M distinct emails, with a failure rate of $< 1.2\%$ and falling. Its robustness is due to the combination of OCaml's type-system and the use of CPS to handle control-flow. The parser combinator library ensures that the full complexity of MIME email can be handled, as well as providing for efficient implementation.

1 Email Parsing is Hard

Email parsing naturally divides into two key components: parsing addresses and parsing bodies (which include headers).

Addresses. Many implementations attempt to parse email addresses using preconceived and often unstated assumptions about what constitutes a valid email address. Often such parsers are implemented using regular expression but this tends to lead to broken address parsing.[1] For example, services often refuse to accept the + character inside the email address, even though this is perfectly valid, used for *tagged emails* and filtering inside the Mail User Agent (MUA). For example, the following are several unusual but valid addresses, all of which MrMime can parse[2]:

```
!#$%&'*+/=?${|}~@iana.org
a(a(b(c)d(e(f))g)h(i)j)@iana.org
+1~1+@iana.org
first.last@[IPv6:111:222:333:444:12:34:56:78]
"first..last"@(that comments are allowed)this.is.ok
```

Bodies. As email became more widely used, people needed to send more complex information than pure text, such as photos, videos, audio. Multipurpose Internet Mail Extensions (MIME) [2] was developed to support this, and describes a recursive format that can include multiple parts, each of which can be encoded with Base64 or Quoted-Printable encodings. Each part has an attached type to avoid any ambiguity about the content of the mail (a precursor of the approach subsequently taken in many other protocols such as HTTP).

This is a example of a valid email, with five parts to be displayed serially: two introductory plain text objects, an embedded multipart message, a `text/enriched` object, and a closing encapsulated text message in a non-ASCII character set. The embedded multipart message itself contains two objects to be displayed in parallel, a picture and an audio fragment.

```
MIME-Version: 1.0
From: Nathaniel Borenstein <nsb@nsb.fv.com>
To: Ned Freed <ned@innosoft.com>
Date: Fri, 07 Oct 1994 16:15:05 -0700 (PDT)
Subject: A multipart example
Content-Type: multipart/mixed;
              boundary=unique-boundary-1

This is the preamble area of a multipart message.
```

[1] E.g., `https://fightingforalostcause.net/content/misc/2006/compare-email-regex.php` compares several such attempts

[2] E.g., `http://oklm-wsh.github.io/Validator/validator.html`

```
--unique-boundary-1

  ... Some text appears here ...

--unique-boundary-1
Content-type: text/plain; charset=US-ASCII

This could have been part of the previous part, but
illustrates explicit versus implicit typing of body parts.

--unique-boundary-1
Content-Type: multipart/parallel; boundary=unique-boundary-2

--unique-boundary-2
Content-Type: audio/basic
Content-Transfer-Encoding: base64

  ... base64-encoded 8000 Hz single-channel
      mu-law-format audio data goes here ...

--unique-boundary-2
Content-Type: image/jpeg
Content-Transfer-Encoding: base64

  ... base64-encoded image data goes here ...

--unique-boundary-2--

--unique-boundary-1
Content-type: text/enriched

This is <bold><italic>enriched.</italic></bold>
<smaller>as defined in RFC 1896</smaller>

Isn't it
<bigger><bigger>cool?</bigger></bigger>

--unique-boundary-1
Content-Type: message/rfc822

From: (mailbox in US-ASCII)
To: (address in US-ASCII)
Subject: (subject in US-ASCII)
Content-Type: Text/plain; charset=ISO-8859-1
Content-Transfer-Encoding: Quoted-printable

  ... Additional text in ISO-8859-1 goes here ...

--unique-boundary-1--
```

2 Emails, Combinators, & CPS

To handle the complexity introduced by the MIME standard, we need to create a parser to process the body of mail *on-the-fly*. This parser must decode the body, encoded with Base64 or Quoted-Printable encoding, according to the `Content-Transfer-Encoding` field; and, if we are inside a *composite* body, the parser must stop at the `boundary` specified by the `Content-Type` field.

To enable this, MrMime is uses parser combinators and CPS style, creating and using ad hoc parsers through composition while processing emails. For example, the following code implements a parser about the body of mail.

This shows three things: (*i*) composition using the disjunction operator `/` between `close_delimiter` and `delimiter` patterns; (*ii*) creation of a tiny parser `stop` that recognises the body delimiter; and (*iii*) generation of a parser for the *multipart* body according to the `Content-Transfer-Encoding` field (and, according to the encoding, uses a specific body parser which can be extended by an user-defined parser).

```
let p_body boundary content =
  let stop =
    let delimiter       = Rfc2046.m_delimiter boundary in
    let close_delimiter = Rfc2046.m_close_delimiter boundary in

    (Rfc2046.p_close_delimiter boundary @ ok ())
    / ((Rfc2046.p_delimiter boundary @ ok ())
       / (fun state -> `Continue state)
       @ (fun () -> roll_back (fun state -> `Stop state) delimiter))
    @ (fun () -> roll_back (fun state -> `Stop state) close_delimiter)
  in

  match Content.encoding content with
  | `Ietf_token _
  | `X_token _
  | `Binary | `Bit8 | `Bit7 -> Rfc5322.p_body stop
  | `Base64                 -> Rfc2045.Base64.p_decode stop
  | `QuotedPrintable        -> Rfc2045.QuotedPrintable.p_decode stop
```

References

[1] D. Crocker. STANDARD FOR THE FORMAT OF ARPA INTERNET TEXT MESSAGES. RFC 822, August 1982.

[2] N. Freed and N. Borenstein. Multipurpose Internet Mail Extensions (MIME) Part One: Format of Internet Message Bodies. RFC 2045, November 1996.