Computerate Specifying

@ Hackathon 112

2021-11-05

Marc Petit-Huguenin

Computerate Specifying

- "Computerate Specifying" original goal is to ensure that examples in RFCs are correct, because examples are often misunderstood as normative.
- A computerate specification is written in AsciiDoc, a variant-less markdown-like format that is extensible by design, and use a xml2rfc v3 renderer provided by the Metanorma project.
- Extensions allow to add code to a document (Literate Programming) and to insert the result of code evaluation in the document itself.

Example Specification

```
> t : String
> t = concat (take rc' (map (\t => show t ++ " ms, ")
>   (transmissions 0 rto)))
>   ++ "and " ++ show (transmission rto rc') ++ " ms"

For example, assuming an RTO of code:[rto]ms, requests would be
sent at times code:[t].
If the client has not received a response
after code:[transmission rto (cast (rc - 1)) + rto * rm] ms, the
client will consider the transaction to have timed out.
```

```
tools computerate -t ietf -x rfc,txt,html,pdf rfc8489.lipkg
```

```
For example, assuming an RTO of 500 ms, requests would be sent at
times 0 ms, 500 ms, 1500 ms, 3500 ms, 7500 ms, 15500 ms, and
31500 ms.  If the client has not received a response after
39500 ms, the client will consider the transaction to have timed
out.
```

Correct By Construction

- We have a tradition of verification tools at the IETF (MIB, ABNF, YANG) but instead computerate specifications allow to generate examples that are already correct by construction.
- That requires to use a programming language that has a type system that can encode higher order logic, which for computerate specifications is Idris2.

ABNF DSL

- The tooling is mostly done, now the work is on providing Idris modules (libraries) that can solve common problems for I-D authors.
- A Domain Specific Language (DSL) can be used to define ABNF grammars:

```
> alpha : Abnf True
> alpha = rule "ALPHA" $ hexRange 0x41 0x5a
>   <|> hexRange 0x61 0x7a

code::[alpha]
```

```
      ALPHA    =  %x41-5A / %x61-7A
```

ABNF Examples

- A discarded idea was to generate examples from an ABNF. This is wrong because not all the constraints in a PDU can be encoded in ABNF:

```
sexp = list / token
token = 1*digit ":" *OCTET
list = "(" sexp ")"
```

- Here the number of OCTET must be equal to the number that precedes the colon, which cannot be expressed in ABNF.

ABNF Examples

- The right way to generate an example is to define a type for sexp, and eventually a DSL:

```
> data SExp : Type where
>   SList : SExp -> SExp -> SExp
>   SToken : String -> SExp

> (>>) = SList
> t = SToken
```

- But how to be sure that a printable value of that type (i.e., an example) is correct according to the ABNF?
- That was the subject of my work at the Hackathon.

Proof of Valid Example

- First we need a way to create a proof that a string is valid according to an Abnf:

```
> data Valid : List Int -> Abnf c -> Type
```

- A proof of that type can be inserted in the AsciiDoc document and will display the validated string.
- Then we just have to implement a conversion between our SExp type and this proof:

```
> example : SExp -> (s : List Int ** Valid s sexp)
```

Convert a Type Instance into a Proof

- Now we can directly insert in the document an example that is correct by construction:

```
code::[snd $ example (do (t "a"); (t "b"); do (T "c"))]
```

- which will display:

```
(1:a1:b(1:c))
```

Links

- Documentation:

https://datatracker.ietf.org/doc/draft-petithuguenin-computerate-specifying/

- The link to the tooling is inside the documentation.
- The feature explained here is part of -15, released on November