

# SPLAT is where it's at!


Konrad Slind  
Collins Aerospace  
CASE Project PI visit  
May 22 2019

# Overview

**SPLAT**<sup>1</sup> is a plugin to the Collins "Architecture-Level Designer Workbench".

**SPLAT** provides support for adding filters between components in order to prevent reception of badly formed or malicious messages.

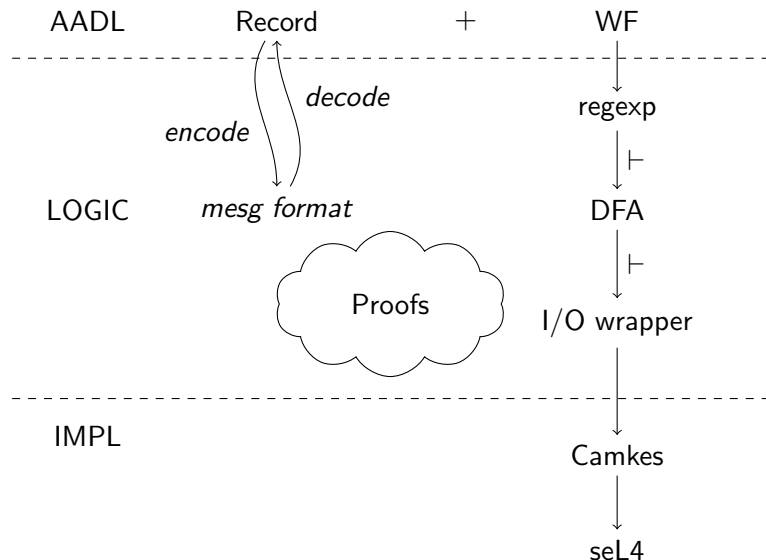
---

<sup>1</sup>SPLAT = *Semantic Properties for Language and Automata Theory* 

# Functionality

- ▶ Input: an AADL architecture annotated with filter specifications
- ▶ Output: a logical theory containing the following elements for each filter specified in the architecture:
  - declaration of the record and enumeration type(s) corresponding to the filter input
  - definition(s) of the wellformedness of the filter input
  - a table-driven DFA that renders a pass-fail verdict on messages, allowing only wellformed messages to pass
  - encoder and decoder mapping between records and the message format
  - theorems showing the correctness of the DFA.

# Picture



# What kinds of records are handled?

A field can have:

- ▶ signed/unsigned numbers
- ▶ constants
- ▶ elements of enumerated types
- ▶ arrays
- ▶ character and string literals
- ▶ sub-records (records of records ...)

Not handled (yet):

- ▶ floating point numbers
- ▶ recursive types (lists, trees, etc)

# Wellformedness

Expressed as constraints on fields

- ▶ numeric fields can be given interval constraints
- ▶ constant fields can directly specified
- ▶ enumeration fields can be restricted to only a subset of the enumeration

Basis of our approach is that interval constraints on numbers can be expressed by regexps.

**NB** Inter-field constraints not allowed (e.g. *field X is the length of field Y* cannot work in **SPLAT**)

# Example

Record type declaration

```
gps = <| lat : int ;  
        lon : int ;  
        alt : int |>
```

Wellformedness

```
good_gps recd <=>  
  -90 <= recd.lat /\ recd.lat <= 90 /\  
  -180 <= recd.lon /\ recd.lon <= 180 /\  
  0 <= recd.alt /\ recd.alt <= 17999
```

# Message format

Wellformedness is a high level predicate, it applies at the level of record datatypes. The message format, however, is what actually gets sent in a communication.

The following dimensions need to be addressed in order to truly pin down the message format:

- ▶ representation: twos complement, sign-magnitude, funky encodings
- ▶ Signed/unsigned numbers
- ▶ Field width
- ▶ Byte order (also bit order)
- ▶ Order of layout of fields
- ▶ Skipped fields
- ▶ Padding, markers, magic numbers, etc
- ▶ Packing
- ▶ ... and probably more



## Example (contd) I

- ▶ Pick an encoding of ints (sign-magnitude), with encoder/decoder **encZ/decZ**.
- ▶ Encoder (autogenerated)

```
encode(gps) =  
  CONCAT [encZ 1 gps.lat;  
          encZ 1 gps.lon;  
          encZ 2 gps.alt]
```

- ▶ Decoder (autogenerated)

```
decode s =  
  case s  
  of [a;b;c;d;e;f;g] =>  
    SOME <| lat := decZ [a;b];  
            lon := decZ [c;d];  
            alt := decZ [e;f;g] |>  
    | otherwise => NONE
```

## Example (contd) II

- ▶ A regexp is autogenerated from the constraints. Each field is described by an *interval regexp* for that field.

- ▶ Regexp (concrete syntax)

```
\i{~90,90}\i{~180,180}\i{0,17999}
```

- ▶ Regexp (AST, prettyprinted)

```
([+] [\000-Z] | [-] [\001-Z])  
([+] [\000-\180] | [-] [\001-\180])  
([+] ([\000-0][F] | .[\000] | .[\001-E]))
```

# Example (contd) III

## ► Regex (AST, raw)

```
Cat
(Or [Cat (Chset (Charset 8796093022208w 0w 0w 0w))
      (Chset (Charset 18446744073709551615w 134217727w 0w 0w));
    Cat (Chset (Charset 35184372088832w 0w 0w 0w))
      (Chset (Charset 18446744073709551614w 134217727w 0w 0w))])

(Cat
  (Or [Cat (Chset (Charset 8796093022208w 0w 0w 0w))
        (Chset (Charset 18446744073709551615w
                      18446744073709551615w 9007199254740991w 0w));
      Cat (Chset (Charset 35184372088832w 0w 0w 0w))
        (Chset (Charset 18446744073709551614w
                      18446744073709551615w 9007199254740991w 0w))])
  (Cat (Chset (Charset 8796093022208w 0w 0w 0w))
    (Or [Cat (Chset (Charset 18446744073709551615w 65535w 0w 0w))
          (Chset (Charset 0w 64w 0w 0w));
        Cat (Chset (Charset 18446744073709551615w
                      18446744073709551615w 18446744073709551615w
                      18446744073709551615w))
          (Chset (Charset 1w 0w 0w 0w));
        Cat (Chset (Charset 18446744073709551615w
                      18446744073709551615w 18446744073709551615w
                      18446744073709551615w))
          (Chset (Charset 18446744073709551614w 63w 0w 0w))])))
```

# Formal properties I

A collection of relationships need to hold between the regexp, the wellformedness condition, the encoder and the decoder:

- ▶ *The record is wellformed iff its corresponding message passes the filter. Formally:*

$$\forall g : gps. \text{good\_gps}(g) \iff \text{encode}(g) \in \mathcal{L}(\text{regexp})$$

- This connects the filter behavior to the high-level property **good\_gps**(*g*) used in compositional reasoning with AGREE.
- L-to-R reading (sender side): if the record is wellformed the corresponding message will be accepted.
- R-to-L reading tells us that **encode** is “good”

## Formal properties II

- *If a message passes the filter then it will decode into a wellformed record.* Formally:

$$\forall s. s \in \mathcal{L}(\text{regexp}) \Rightarrow \text{good\_gps}(\text{valOf}(\text{decode}(s)))$$

- This is a *receiver-side* property
- The checking of the message for wellformedness is accomplished by the filter, before the decoder gets called.
- Maybe be possible to strengthen this to an iff, but not sure it's useful.

# Formal properties III

- ▶ *decode of encode is the identity.* Formally:

$$\forall g. \mathbf{decode}(\mathbf{encode}(g)) = \mathbf{SOME}(g)$$

We attempt to prove this property, although it is not always true (e.g. when some fields are superfluous to a message and the encoder omits them).

- ▶ *encode is an injection (1-1 function).* In other words, encoding doesn't map two different records to each other. We don't (yet) try to prove this property. It has the same issues as the round-trip property: dropping superfluous fields can result in identical messages.

# Encode and decode

Note that **encode** and **decode** are present in order to express the desired properties, especially the first one. So they can be regarded as artifacts primarily used to state correctness.

We could also generate implementations, e.g. in CakeML, for **encode/decode**. It would be nice to generate versions in popular languages for embedded systems.

We could use the logic versions of **encode/decode** as specifications against which to prove correctness of implementations of **encode/decode** by others in standard PLs.

# Discussion I

- ▶ **SPLAT** deliberately takes a formal language approach, in which we try to apply formal language technology (e.g. regexps, grammars, DFAs and DPDAs).
- ▶ This has benefits and limitations. The main limitation is that the functionality of some (many?) filters cannot be captured by a regular expression or with a grammar.
- ▶ A common question is *“Why bother? Can’t you just write (or generate) the relatively simple validity checkers and verify them?”*



# Discussion II

Response:

- **SPLAT** filters run exclusively at the bitstring level. There is no interpretation of data. Seems that this should lead to fast and inherently uniform execution.
- The validity checker approach requires (essentially) parsing messages into records and then running the validity checks.
- For complex messages the validity checker may itself be complex.
- For long messages, the input need not be stored in memory. State machines go character-by-character, so they need very little space (other than to store their static tables, which may be large).

# Future work

- ▶ Ongoing discussion with colleagues on adopting (or creating) an intermediate language for expressing message formats
- ▶ Performance and scaling
- ▶ Extend **SPLAT** to CFLs and produce filters for complex tree-shaped data
- ▶ Extend to handle features not supported by formal language technology (e.g. length fields)