Splitting Sail definitions

Sail source

```
default Order dec
$include <prelude.sail>

val rX : bits(5) -> bits(32)

val wX : (bits(5), bits(32)) -> unit

overload X = {rX, wX}

enum Op = ADD | SUB

$[split op]
function instr(rd: bits(5), rs1: bits(5), rs2: bits(5), op: Op) -> unit = {
    let rs1_val = X(rs1);
    let rs2_val = X(rs2);

    let result: bits(32) = match op {
        ADD => add_bits(rs1_val, rs2_val),
        SUB => sub_bits(rs1_val, rs2_val),
    };

    X(rd) = result
}
```

Result

Sometimes we have a Sail function that corresponds to multiple functions we want to document. Here we can split the function by applying *constant propagation* with the `split` attribute in Sail. This works when the function has an enumeration as an argument.

WARNING

This feature is somewhat experimental as it relies on calling Sail's constant propagation pass and pretty printer during document bundle preparation, neither of which were really intended for this use case, so while it works for simple functions you might run into places where it fails for more complex inputs. Notice also that this happens after overloads have been resolved, so we see `rX` and `wX` in the below examples, rather than the overload `X`.

As an example, here we can take the above `instr` which implements both `ADD` and `SUB` and generate just the `ADD` case:

```
sail::instr[split=ADD]
```

which produces:

```
let rs1_val = rX(rs1);
let rs2_val = rX(rs2);
let result : bits(32) = add_bits(rs1_val, rs2_val);
wX(rd, result)
```

Alternatively, for the `SUB` case:

```
sail::instr[split=SUB]
```

produces

```
let rs1_val = rX(rs1);
let rs2_val = rX(rs2);
let result : bits(32) = sub_bits(rs1_val, rs2_val);
wX(rd, result)
```