Documenting definitions with multiple clauses

Sail source

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

/* Pretend we have accessors for reading and writing registers */
val rX : bits(5) -> bits(32)

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

overload X = {rX, wX}

scattered union Instr

val execute : Instr -> unit

union clause Instr = Add : (bits(5), bits(5), bits(5))

function clause execute Add(rd, rx, ry) = {
    X(rd) = add_bits(X(rx), X(ry))
}

union clause Instr = Sub : (bits(5), bits(5), bits(5))

function clause execute Sub(0b00000, rx, ry) = {
    ()
}

function clause execute Sub(rd, rx, ry) = {
    X(rd) = sub_bits(X(rx), X(ry))
}
```

Result

The execute function is a *scattered function*, a Sail feature that allows us to split apart the various cases of the function into multiple *clauses*. It may seem hard to document these, as the function clauses share the same name, and are only distinguished by their *pattern*. To include just the `Add` clause, we can use the following command:

```
sail::execute[clause="Add(_, _, _)"]
```

which produces:

```
function clause execute Add(rd, rx, ry) = {
    X(rd) = add_bits(X(rx), X(ry))
}
```

The `clause` attribute allows us to match on the pattern, using syntax similar to that found in Sail. The underscore is the *wildcard* pattern, that allows us to match anything.

The `Sub` instruction has two function clauses. For the first one where the destination register is `0b00000` we can include it using:

```
sail::execute[clause="Sub(0b00000, _, _)"]
```

which produces:

```
function clause execute Sub(0b00000, rx, ry) = {
    ()
}
```

The next clause we can include similarly, like so:

```
sail::execute[clause="Sub(rd, _, _)"]
```

which produces:

```
function clause execute Sub(rd, rx, ry) = {
    X(rd) = sub_bits(X(rx), X(ry))
}
```