1. Bugs in Renee_Value

1.1 overflow of vector-bit operations

We suppose, in the following operations, when the second operand *b* is FALSE, then the second element of result would be FALSE, which means that there is no overflow in the operation. However, this is not the situation in sail_values.ml(commit version ce962ff), the second element of their result would be TRUE when translated to PVS. We found 3 bugs of this kind. In a later version of sail_values.ml, they have modified the bug.

```
add_overflow_vec_bit_signed(bv, b) : MACRO [bvec[N], bit, bit] =
  IF NOT b
  THEN (bv, FALSE, FALSE)
  ELSE
    LET add_result = signed(bv) + 1 IN
    LET overflow = add_result > maxint[N] OR add_result < minint[N],
      c_out = (mod(floor((unsigned[N](bv) + 1)/exp2(N)), 2) /= 0) IN
    (to_vec_inc(N, add_result), overflow , c_out)
  ENDIF
```

1.2

```
minus_overflow_vec_bit(bv, b) : MACRO [bvec[N], bit, bit] =
  IF NOT b
  THEN (bv, FALSE, FALSE)
  ELSE
    LET minus_result = unsigned[N](bv) - 1 IN
    LET overflow = minus_result > exp2(N) OR minus_result < 0,
      c_out = (mod(floor(minus_result/exp2(N)), 2) /= 0) IN
    (to_vec_inc(N, minus_result), overflow, c_out)
  ENDIF
```

1.3

```
minus_overflow_vec_bit_signed(bv, b) : MACRO [bvec[N], bit, bit] =
      IF NOT b
      THEN (bv, FALSE, FALSE)
      ELSE
             LET minus_result = signed(bv) - 1 IN
             LET overflow = minus_result > maxint[N] OR minus_result < minint[N],
                    c_out = (mod(floor((unsigned[N](bv) - 1)/exp2(N)), 2) /= 0) IN
             (to_vec_inc(N, minus_result), overflow, c_out)
      ENDIF
```

1.4 *mod* operation

Previously, we directly use the *mod* definition in PVS, which can be represented as:

```
mod(n:int, m:nzint) : int = n - m * floor(n/m)
```

However, the mod definition in OCaml is different, thus we define a new modulo function as:

```
modulo(n:int, m:nzint) : MACRO int =
  IF (n > 0 AND m < 0) OR (n < 0 AND m > 0)
  THEN n - m * ceiling(n/m)
  ELSE n - m * floor(n/m)
```

```
    ENDIF
```

1.5 *mod_signed* operation

In *mult_vec_range_signed* definition, we previously used the original *mod* operation. However, we figure out that this does not satisfy the definition in sail_values.ml, thus we redefine a *mod_signed* operation and replace mod with mod_signed.

Previous version:

```
mult_vec_range_signed(bv, m) : MACRO bvec[2*N] = to_vec_inc(2*N, signed(bv) * mod(m, exp2(N)))
```

Latest version:

```
mod_signed(m:int, n:posnat): MACRO int =
   LET mod_result = mod(m, exp2(n)) IN
   IF mod_result < exp2(n - 1)
   THEN mod_result
   ELSE mod_result - exp2(n)
   ENDIF
mult_vec_range_signed(bv, m) : MACRO bvec[2*N] = to_vec_inc(2*N, signed(bv) *
mod_signed(m, N))
```

1.6 *quot* operation

In a previous version of *quot_overflow_vec*, we suppose that if the second operand can be converted to 0, then the *quot_overflow_vec* operation does not work.

Thus we declare it as

```
quot_overflow_vec(bv1:bvec[N], bv2:{bv:bvec[M]| unsigned[M](bv) /= 0})
```

However, in sail_values.ml, the quot_overflow_vec does not have this limitation. They can handle the situation that the second operand can be translated to 0. So we modified our definition.

```
quot_overflow_vec(bv1:bvec[N], bv2:bvec[M]) : MACRO [bvec[N], bit, bit] =
   IF unsigned(bv2) = 0
   THEN (bvec0[N], FALSE, FALSE)
   ELSE
      LET quot_result = floor(unsigned[N](bv1)/unsigned[M](bv2)) IN
      LET overflow = quot_result > exp2(M),
         c_out = (mod(floor(quot_result/exp2(N)), 2) /= 0) IN
      (to_vec_inc(N, quot_result), overflow , c_out)
   ENDIF
```

2. Bugs in Renee_Basic

Since Renee_Basic is still in development, there might still exist some bugs. The already fixed ones are listed as follows.

2.1 double-quote

When we first generate a string, we use "\"" to show a double quotation mark inside the string. However, PVS cannot recognize this kind of notation, we have to switch to a constant, named **doublequote**, to replace all the "\"" inside the string. That means, the string has to be split to separate parts based on the location of "\"", and then we use doublequote to replace the "\"". Finally, the separate parts are combined together to a new string using **o** infix operator.

2.2 empty list

The representation of empty list (::) will cause errors since PVS cannot recognize the type of the list element. We have to use the traditional representation like **null[bool]**.

2.3 variable element in *list*

If the elements in a list representation are variables, like x, y, z. Then the shortcut representation *(: x, y, z :)* is not available, which means we have to switch to *cons(x, cons(y, cons(z, null)))*

2.4 backslash

PVS cannot recognize the backslash representation as "\\", it will give an error message as (Extra "). We have to use the special representation, like **charcode(92)**

2.5 char representation

The char in PVS cannot be represented as 'a'. We have to use char(97) or "a"(0) to get the right representation.