#### Code and Specification

For the next section we introduced the '*div2*' method which simply returned a value divided by two. No additional alterations were required for this method. We then introduced the leftMost method which.....

#### Verification

ESC Verification of the div2 method resulted in an (error) for Line .., however there appeared no issue with this clause and in the resulting SC process all clauses were deemed valid.

The 'leftMost' method had an ArithmeticOperationRange error during initial ESC verification checks with Line … having no upper bound resulting in the upper bound of the integer type being broken after the multiplication operation on Line... Restricting Line … to have an upper bound of Integer.MAC\_VALUE / 2 solved the issue.

#### Analysis

The ESC verification of the 'div' and 'leftMost' methods was the easiest in the program. The (error) for the 'div' method was worrying behaviour since it was returning invalid results for specifications that were valid which may potentially mean there is a flaw in the proof system and invalid results could being returned valid. As this was the only occurrence of this type of action happening, we believed it was an anomaly but still should be noted.

The 'leftMost' methods upper bound issue was easy to spot and initially fix with an upper bound of 1000, and later, after conversations with David R.Cok on other similar issues, resolved to using the Integer.MAX\_VALUE variable.

#### Code and Specification

We were making good progress up to this point with the majority of the KeY specifications being usable in the OpenJML, and we moved forward introducing the 'pow2' method accordingly. This was implemented as a recursive method to return the 2 to the power of the variable 'x' passed in by the user.

The recursive implementation was replaced with an interative version after it became clear during verification that the recursive method employed by the KeY developers did not work in the same fashion using OpenJML. As an example, we set the variable 'x' to be 3 which should have returned the value 8 but instead returned a value 2147483646 which is the Integer.MAX\_VALUE - 1 value as can be seen from the trace in Figure (xxx). It was discovered that recursion is not fully built-in to the tool yet with the measured\_by clause not implemented to ensure termination of the recursive method, therefore we introduced the iterative implementation.

*TRACE of q1\_2012.PrefixSumRec.pow2(int)*

*/home/eo37/workspace/Prefix\_Sum/src/q1\_2012/PrefixSumRec.java:55: requires x >= 0 && x < 5;*

*VALUE: x === 3*

*VALUE: 0 === 0*

*VALUE: x >= 0 === true*

*VALUE: x === 3*

*VALUE: 5 === 5*

*VALUE: x < 5 === true*

*VALUE: x >= 0 && x < 5 === true*

*/home/eo37/workspace/Prefix\_Sum/src/q1\_2012/PrefixSumRec.java:62: return x == 0 ? 1 : 2 \* pow2(x - 1);*

*VALUE: x === 3*

*VALUE: 0 === 0*

*VALUE: x == 0 === false*

*VALUE: 2 === 2*

*VALUE: x === 3*

*VALUE: 1 === 1*

*VALUE: x - 1 === 2*

*VALUE: pow2(x - 1) === 2147483646*

*VALUE: 2 \* pow2(x - 1) === 4294967292*

*VALUE: x == 0 ? 1 : 2 \* pow2(x - 1) === 4294967292*

*/home/eo37/workspace/Prefix\_Sum/src/q1\_2012/PrefixSumRec.java:62: ArithmeticOperationRange assertion: !(0 < x && 1 < 0) || x <= 2147483647 + 1*

*VALUE: !(0 < x\_1543\_0\_\_\_1 && 1 < 0) || x\_1543\_0\_\_\_1 <= 2147483647 + 1 === true*

*/home/eo37/workspace/Prefix\_Sum/src/q1\_2012/PrefixSumRec.java:62: ArithmeticOperationRange assertion: !(x < 0 && 0 < 1) || -2147483648 + 1 <= x*

*VALUE: !(x\_1543\_0\_\_\_1 < 0 && 0 < 1) || -2147483648 + 1 <= x\_1543\_0\_\_\_1 === true*

*/home/eo37/workspace/Prefix\_Sum/src/q1\_2012/PrefixSumRec.java:55: Precondition assertion: \_$CPRE\_\_8*

*/home/eo37/workspace/Prefix\_Sum/src/q1\_2012/PrefixSumRec.java:62: ArithmeticOperationRange assertion: -9223372036854775808L <= (long)2 \* \_JML\_\_tmp141 && (long)2 \* \_JML\_\_tmp141 <= 9223372036854775807L*

*VALUE: -9223372036854775808L <= (long)2 \* \_JML\_\_tmp141 && (long)2 \* \_JML\_\_tmp141 <= 9223372036854775807L === true*

*/home/eo37/workspace/Prefix\_Sum/src/q1\_2012/PrefixSumRec.java:57: ensures \result > 0 && \result < Integer.MAX\_VALUE;*

*VALUE: \result === 2147483646*

*VALUE: 0 === 0*

*VALUE: \result > 0 === true*

*VALUE: \result === 2147483646*

*VALUE: Integer.MAX\_VALUE === 2147483647*

*VALUE: \result < Integer.MAX\_VALUE === true*

*VALUE: \result > 0 && \result < Integer.MAX\_VALUE === true*

*/home/eo37/workspace/Prefix\_Sum/src/q1\_2012/PrefixSumRec.java:62: Invalid assertion (Postcondition)*

*: /home/eo37/workspace/Prefix\_Sum/src/q1\_2012/PrefixSumRec.java:57: Associated location*

#### Verification

ESC Verification first determined, as stated above, that the recursive implementation along with its specification were not going to work with OpenJML so we used an iterative implementation. The verification of this approach however, also brought up a lot of issues. The first of which was on Line … , specifically the \product quantifier. This product quantifer should have returned a result a result to match 2 to the power of 'x' but was instead returning seemingly random values, many of which were not even multiples of 2 or were below 0 despite a precondition stating 'x' must be equal to or greater than 0.

Another issue was that the variables uses in the method appeared to be not holding their values in memory and would change from line to line. The count variable was set to 1 on entering the method and if 'x' is 0, the returned value should still be 1, however the resulting value was returning 2147483647 (Integer.MAX\_VALUE). Another example of this was when 'x' was 0 but value changed by the time the while loop came so it executed the operation on count returning 1073741824.

#### Analysis

All of the OpenJML issues combined to make the verification of the 'pow2' method almost impossible and resulted in mass amounts of time being wasted in pursuit of these issues and determining a resolution. It was later learned from David that although the \product quantifer is used in OpenJML, its implementation on the back-end has not been completed so is running an unknown process returning random values.

The memory issues are a far greater problem and have been addressed to David for resolution. The investigation into these issues is still ongoing as of writing this report, with no resolution available as of yet. As a result, all future methods that wither relied on the specification or implementation of the 'pow2' method could no longer be adequately proven. This memory issue in effect proved terminal as all future methods use 'pow2' directly or indirectly within their specifications resulting in the end of the verification process for the PrefixSum algorithm.

TRACE of q1\_2012.PrefixSumRec.pow2(int)

/home/eo37/workspace/Prefix\_Sum/src/q1\_2012/PrefixSumRec.java:56: requires x == 0;

VALUE: x === 0

VALUE: 0 === 0

VALUE: x == 0 === true

/home/eo37/workspace/Prefix\_Sum/src/q1\_2012/PrefixSumRec.java:65: count = 1

VALUE: 1 === 1

VALUE: count = 1 === 1

/home/eo37/workspace/Prefix\_Sum/src/q1\_2012/PrefixSumRec.java:67: //@ loop\_invariant count >= 1 && x >= 0;

VALUE: count === 1

VALUE: 1 === 1

VALUE: count >= 1 === true

VALUE: x === 0

VALUE: 0 === 0

VALUE: x >= 0 === true

/home/eo37/workspace/Prefix\_Sum/src/q1\_2012/PrefixSumRec.java:67: LoopInvariantBeforeLoop assertion: \_JML\_\_conditionalResult\_135

/home/eo37/workspace/Prefix\_Sum/src/q1\_2012/PrefixSumRec.java:67: //@ loop\_invariant count >= 1 && x >= 0;

VALUE: count === 1073741824

VALUE: 1 === 1

VALUE: count >= 1 === true

VALUE: x === 1

VALUE: 0 === 0

VALUE: x >= 0 === true

VALUE: count >= 1 && x >= 0 === true

/home/eo37/workspace/Prefix\_Sum/src/q1\_2012/PrefixSumRec.java:69: Loop test

VALUE: x === 1

VALUE: 0 === 0

VALUE: x > 0 === true

VALUE: (x > 0) === true

/home/eo37/workspace/Prefix\_Sum/src/q1\_2012/PrefixSumRec.java:71: count += count

VALUE: count === 1073741824

VALUE: count += count === 0

/home/eo37/workspace/Prefix\_Sum/src/q1\_2012/PrefixSumRec.java:71: ArithmeticOperationRange assertion: !(0 < q1\_2012.PrefixSumRec.count && 0 < q1\_2012.PrefixSumRec.count) || q1\_2012.PrefixSumRec.count <= 2147483647 - q1\_2012.PrefixSumRec.count

VALUE: !(0 < count\_138\_1722\_\_\_21 && 0 < count\_138\_1722\_\_\_21) || count\_138\_1722\_\_\_21 <= 2147483647 - count\_138\_1722\_\_\_21 === false

/home/eo37/workspace/Prefix\_Sum/src/q1\_2012/PrefixSumRec.java:71: Invalid assertion (ArithmeticOperationRange)

TRACE of q1\_2012.PrefixSumRec.pow2(int)

/home/eo37/workspace/Prefix\_Sum/src/q1\_2012/PrefixSumRec.java:56: requires x == 0;

VALUE: x === 0

VALUE: 0 === 0

VALUE: x == 0 === true

/home/eo37/workspace/Prefix\_Sum/src/q1\_2012/PrefixSumRec.java:65: count = 1

VALUE: 1 === 1

VALUE: count = 1 === 1

/home/eo37/workspace/Prefix\_Sum/src/q1\_2012/PrefixSumRec.java:67: //@ loop\_invariant count >= 1 && x >= 0;

VALUE: count === 1

VALUE: 1 === 1

VALUE: count >= 1 === true

VALUE: x === 0

VALUE: 0 === 0

VALUE: x >= 0 === true

/home/eo37/workspace/Prefix\_Sum/src/q1\_2012/PrefixSumRec.java:67: LoopInvariantBeforeLoop assertion: \_JML\_\_conditionalResult\_135

/home/eo37/workspace/Prefix\_Sum/src/q1\_2012/PrefixSumRec.java:67: //@ loop\_invariant count >= 1 && x >= 0;

VALUE: count === 2147483647

VALUE: 1 === 1

VALUE: count >= 1 === true

VALUE: x === 0

VALUE: 0 === 0

VALUE: x >= 0 === true

VALUE: count >= 1 && x >= 0 === true

/home/eo37/workspace/Prefix\_Sum/src/q1\_2012/PrefixSumRec.java:69: Loop test

VALUE: x === 0

VALUE: 0 === 0

VALUE: x > 0 === false

VALUE: (x > 0) === false

/home/eo37/workspace/Prefix\_Sum/src/q1\_2012/PrefixSumRec.java:74: return count;

VALUE: count === 2147483647

/home/eo37/workspace/Prefix\_Sum/src/q1\_2012/PrefixSumRec.java:58: ensures \result > 0 && \result < Integer.MAX\_VALUE;

VALUE: \result === 2147483647

VALUE: 0 === 0

VALUE: \result > 0 === true

VALUE: \result === 2147483647

VALUE: Integer.MAX\_VALUE === 2147483647

VALUE: \result < Integer.MAX\_VALUE === false

/home/eo37/workspace/Prefix\_Sum/src/q1\_2012/PrefixSumRec.java:74: Invalid assertion (Postcondition)

: /home/eo37/workspace/Prefix\_Sum/src/q1\_2012/PrefixSumRec.java:58: Associated location