Examples for Parity and Hamming Refinement Patterns

A. Iliasov and A. Romanovsky

1 Example

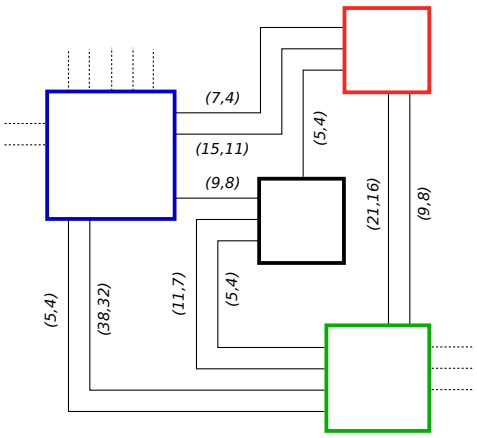


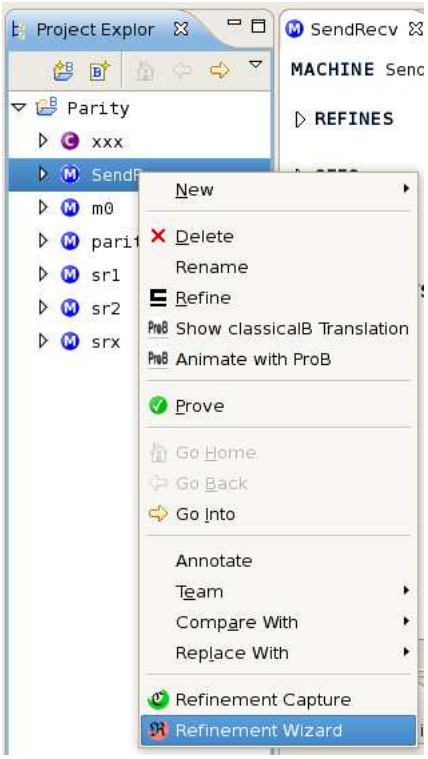
Figure 1: A distributed system a with a large number of interconnection. Each connection may have its own reliability properties and requirements and thus different algorithms must be used. Manually implementing an error detection or error correction algorithm for each connection is not viable.

We consider a design of a distributed system with a large number of not completely reliable interconnections 1. The parity and hamming patterns are used to construct a development with receiver and sender logic that can tolerate a single bit change in a transmitted word. The parity pattern makes its possible to detect that an error was introduced in transmission. The receiver in this case might try to request sender to resend the same data. The hamming parity handles single bit changes transparently. Not only it detects such situation, but also corrects the transmitted word using the addition parity bits added to the original word.

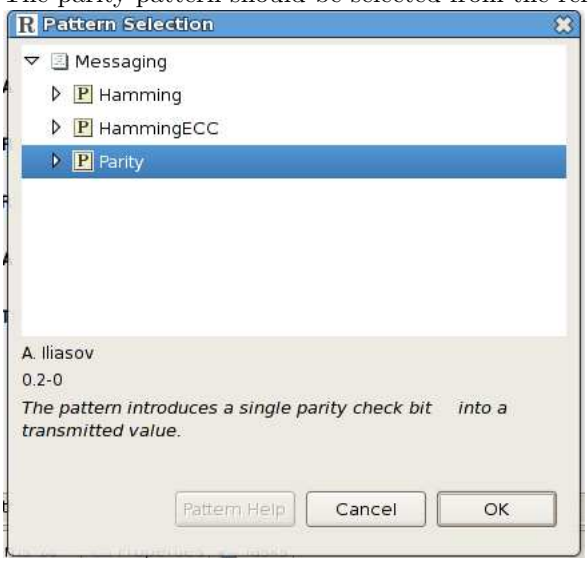
This document only includes Event-B specification showing the results of application of the two pattern to a simple model presented below.

1.1 Recreating the models

These models can be trivially recreated in the RODIN Platform using the patterns. This required downloading the abstract model or entering it into the platform and also downloading the two patterns (with the *Import* option in the *Patterns* menu). The Event-B model with parity check algorithm is constructed by applying the parity pattern to the abstract model. For this, click on the abstract model in the Project Explorer view



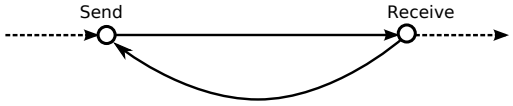
The parity pattern should be selected from the refinement pattens dialog



The configuration for this example is automatically detected by the pattern and not customisation is required. After several clicks on the "Apply" button, the refinement wizard produces a new model. To see how the patterns react on the changes in the input model try different typing predicates for the channel variable in the abstract model. Also try adding additional events or changing the *sender* and *receiver* events in the abstract model.

1.2 Communication Loop

In its simplest, a communication loop is modelled by allocating a shared variable for a communication channel. To indicate that a channel has data to be read, the sender raises a ready flag. In its turn, the receiver, after reading from the channel, resets the flag to allow the sender to send new data. The process is repeated until the sender does not have any data to send or the receiver does not want to read the incoming data anymore



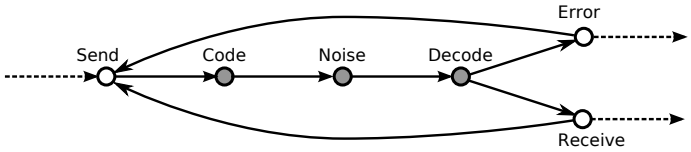
system *SendRecv*
variables
$ch, outv, rd$
invariant
$ch \in 0 \mathinner{..} 15$
$outv \in 0 \mathinner{..} 15$
$rd \in BOOL$
initialisation
$ch := 0$
$outv := 0$
$rd := TRUE$

receive = **when**
$rd = FALSE$
then
$outv := ch$
$rd := TRUE$
end

send = **any** v **where**
$rd = TRUE \wedge v \in 0 \mathinner{..} 15$
then
$ch := v$
$rd := FALSE$
end

1.3 Introducing Error Detection

To cope with unreliable communication channels we add redundant information to each communication message. The simplest form of this approach is to add a parity bit that is set in such a way that sum of all bits is even (or odd, but this must be agreed upon beforehand). Single parity bit can detect single error in transmission. It is likely to give false positives if more than one error occurs.



The new variables are

ch_ph, ch_err, ch_b

$ch\_ph \in 0 \mathinner{..} 4$
$ch\_err \in BOOL$
$ch\_b \in 1 \mathinner{..} 5 \rightarrow \{0, 1\}$

$ch\_ph := 0$
$ch\_err := false$
$ch\_b := 1 \mathinner{..} 5 \times \{0\}$

The *ch_ph* variable controls the order in which the new events are enabled. It also prevents looping of the new events

send	$ch\_ph = 0$
code	$ch\_ph = 1$
noise	$ch\_ph = 2$
decode	$ch\_ph = 3$
receive and *ch_error*	$ch\_ph = 4$

Variable *ch_err* indicates a detected error in the transmitted value

receive	$ch\_err = false$
ch_error	$ch\_err = true$

Variable *ch_b* is a bitwise representation of the abstract communication channel variable. It contains a parity check bit which we choose to put at the last position in the transmitted word. Below are some example of the codewords in the parity check bit algorithm

decimal	binary coding with parity bit
0	0 0 0 0 0
1	1 0 0 0 1
2	0 1 0 0 1
3	1 1 0 0 0
15	1 1 1 1 0

The coding event, constructed by the parity pattern, is the following

ch_code = **any** $p, b1, b2, b3, b4$ **where**
$p \in \{0,1\} \wedge b1 \in \{0,1\} \wedge b2 \in \{0,1\} \wedge b3 \in \{0,1\} \wedge b4 \in \{0,1\} \wedge$
$ch\_ph = 1 \wedge$
$ch = 1 * b1 + 2 * b2 + 4 * b3 + 8 * b4 \wedge$
$p = (b1 + b2 + b3 + b4) \bmod 2$
then
$ch\_b := \{1 \mapsto b1\} \cup \{2 \mapsto b2\} \cup \{3 \mapsto b3\} \cup \{4 \mapsto b4\} \cup \{5 \mapsto p\}$
$ch\_ph := 2$
end

We use separate variables for each bit instead of function as it is a more friendly approach both for the ProB animator and the Platform theorem prover.

The invariant linking abstract channel variable and the new channel variable.

$$ch\_ph = 2 \Rightarrow \left(\begin{array}{l} ch = ch\_b(1) + 2 * ch\_b(2) + 4 * ch\_b(3) + 8 * ch\_b(4) \wedge \\ (ch\_b(1) + ch\_b(2) + ch\_b(3) + ch\_b(4)) \bmod 2 = ch\_b(5) \end{array} \right)$$

The event introducing errors. Sometimes it "fails" to distort transmission is does not analyse the value it changes, so it could be that nothing is changed after all

ch_noise = **any** b, v **where**
$ch\_ph = 2 \wedge b \in 1 \,..\, 5 \wedge v \in \{0,1\}$
then
$ch\_b := ch\_b \Leftarrow \{b \mapsto v\}$
$ch\_ph := 3$
end

The invariant says that the abstract channel variable and the new one are different by one bit at most. This invariant was important to analyse the pattern correctness but it not very useful for the theorem prover

$$ch\_ph = 3 \Rightarrow \exists \{k \mapsto x\} \cdot \left(\begin{array}{l} k \in dom(b) \wedge x \in \{0,1\} \wedge \\ ch = \sum_{j \in 1..4} 2^{j-1} * b'(j) \wedge \\ \sum_{j \in 1..4} b'(j) = b'(5) \end{array} \right)$$

where $b' = b \Leftarrow \{k \mapsto x\}$

The decoding event raises the error if the parity check fails

ch_decode = **any** par **where**
$ch\_ph = 3 \wedge par \in \{0,1\} \wedge$
$par = (ch\_b(1) + ch\_b(2) + ch\_b(3) + ch\_b(4) + ch\_b(5)) \bmod 2$
then
$ch\_err := bool(par \neq 0)$
$ch\_ph := 4$
end

And here is the final invariant which says if there were no error in transmission 9according to the parity check algorithm) then the received value is the sent value

$$ch\_ph = 4 \wedge ch\_err = false \Rightarrow ch = ch\_b(1) + 2 * ch\_b(2) + 4 * ch\_b(3) + 8 * ch\_b(4)$$

The abstract receiver event is split to handle the case when transmission fails

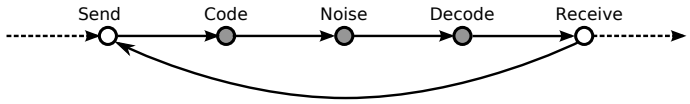
ch_error = **when**
$rd = false \wedge ch\_ph = 4 \wedge ch\_err = true$
then
$outv := ch$
$rd := true$
$ch\_ph := 0$
end

1.4 Detection and Correction of Single Error with a Hamming Code

Since our noise event flips only one bit, we can do much better than simply adding a parity check. An single error correcting code, such as hamming can tolerate change in a single bit of a transmitted word by adding several check

bits. The general idea behind hamming codes is placing enough check bits in such manner that it possible to find which bit was flipped. The table below contains example of hamming code words, the check bits are underlined

decimal	hamming coding (7, 4)
0	$\underline{0}\ \underline{0}\ 0\ \underline{0}\ 0\ 0\ 0$
1	$\underline{1}\ \underline{1}\ 1\ \underline{0}\ 0\ 0\ 0$
2	$\underline{1}\ \underline{0}\ 0\ \underline{1}\ 1\ 0\ 0$
3	$\underline{1}\ \underline{1}\ 1\ \underline{1}\ 1\ 0\ 0$
15	$\underline{1}\ \underline{1}\ 1\ \underline{1}\ 1\ 1\ 1$



The coding event computes all the parity bits and also converts the sent value into the binary form

ch_code = **any** $b3, b5, b6, b7$ **where**
$ch\_ph = 1 \wedge$
$ch = 1 * b3 + 2 * b5 + 4 * b6 + 8 * b7 \wedge$
$b3 \in \{0,1\} \wedge b5 \in \{0,1\} \wedge b6 \in \{0,1\} \wedge b7 \in \{0,1\}$
then
$ch\_b := \{3 \mapsto b3\} \cup \{5 \mapsto b5\} \cup \{6 \mapsto b6\} \cup \{7 \mapsto b7\} \cup$
$(\{1 \mapsto ((b3 + b5 + b7) \bmod 2)\}) \cup$
$(\{2 \mapsto ((b3 + b6 + b7) \bmod 2)\}) \cup$
$(\{4 \mapsto ((b5 + b6 + b7) \bmod 2)\})$
$ch\_ph := 2$
end

The gluing invariant, similar the one we used for the parity pattern. It says how the value of the word sent in the channel with hamming pattern encoding can be decoded back

$$ch\_ph = 2 \Rightarrow ch = ch\_b(3) + 2 * ch\_b(5) + 4 * ch\_b(6) + 8 * ch\_b(7)$$

The decoding event computes all the parity bits and corrects the channel value if an error detected

ch_decode = **any** $par1, par2, par3$ **where**
$(ch\_ph = 3) \wedge$
$(par1 = ((ch\_b(1) + ch\_b(3) + ch\_b(5) + ch\_b(7)) \bmod 2)) \wedge$
$(par2 = ((ch\_b(2) + ch\_b(3) + ch\_b(6) + ch\_b(7)) \bmod 2)) \wedge$
$(par3 = ((ch\_b(4) + ch\_b(5) + ch\_b(6) + ch\_b(7)) \bmod 2)) \wedge$
$(par1 \neq 0 \vee par2 \neq 0 \vee par3 \neq 0) \wedge$
$par1 \in \{0,1\} \wedge par2 \in \{0,1\} \wedge par3 \in \{0,1\}$
then
$ch\_b := ch\_b \Leftarrow (\{1 * par1 + 2 * par2 + 4 * par3 \mapsto$
$(((ch\_b(1 * par1 + 2 * par2 + 4 * par3)) + 1) \bmod 2)\})$
$ch\_ph := 4$
end

The transmission was successful if all the parity bits are even

$$ch\_ph = 4 \wedge ch\_err = false \Rightarrow \left(\begin{array}{l} ch = ch\_b(3) + 2 * ch\_b(5) + 4 * ch\_b(6) + 8 * ch\_b(7) \\ ch\_b(1) + ch\_b(3) + ch\_b(5) + ch\_b(7) = 0 \\ ch\_b(2) + ch\_b(3) + ch\_b(6) + ch\_b(7) = 0 \\ ch\_b(4) + ch\_b(5) + ch\_b(6) + ch\_b(7) = 0 \end{array} \right)$$

Pass-through cares for the case when no bits were flipped

$ch\_nodecode$ = **any** $par1, par2, par3$ **where**
$ch\_ph = 3 \wedge$
$par1 = (ch\_b(1) + ch\_b(3) + ch\_b(5) + ch\_b(7)) \bmod 2 \wedge$
$par2 = (ch\_b(2) + ch\_b(3) + ch\_b(6) + ch\_b(7)) \bmod 2 \wedge$
$par3 = (ch\_b(4) + ch\_b(5) + ch\_b(6) + ch\_b(7)) \bmod 2 \wedge$
$par1 = 0 \wedge par2 = 0 \wedge par3 = 0) \wedge par1 \in \{0,1\} \wedge par2 \in \{0,1\} \wedge par3 \in \{0,1\}$
then
$ch\_ph := 4$
end

This invariant states that we can always correct a a distorted value using the redundant check bits. It is crucial for proving correctness of the decoding event

$$ch\_ph = 3 \Rightarrow \exists\{k \mapsto x\} \cdot \left(\begin{array}{l} k \in dom(b) \wedge x \in \{0,1\} \wedge \\ ch = \sum_{j \in \{3,5,6,7\}} 2^{j-1} * b'(j) \wedge \\ \sum_{j \in \{1,3,5,7\}} b'(j) = 0 \\ \sum_{j \in \{2,3,6,7\}} b'(j) = 0 \\ \sum_{j \in \{4,5,6,7\}} b'(j) = 0 \end{array} \right)$$

where $b' = b \lhd\!\!\!+ \{k \mapsto x\}$