Prep Exercises and Vocareum Submissions

- Must run your model locally first. Do not submit if your model is not working locally (especially if it has syntax errors.)
- If there are syntax errors on Vocareum with REF checks, it means you are not complying with the requirements (naming of models, channels, global variables, definition of required macros, etc.)
- Remember:
  - The solution is never unique!
  - You may pass the sanity checks and still have the wrong model!

Tiny Promela Exercises

- For each exercise:
- Create a Promela model
- Run a random simulation, visualize what's going on
- Run a safety verification with these options on:
  - Safety (state properties)
    - Assertion violations
    - Invalid end states
    - Exhaustive search
- Does the verification fail? Why?
- If fails, run simulation in guided mode to replay the error trace to figure out why

Model1.pml

- Model a simple system consisting of a `Sender` and `Receiver` process communicating via a channel `s2r`
- `s2r` has a capacity of two slots; each slot can store just a single bit
- `Sender` sends two bits, 0 and then 1, one after the other, on the channel and stops
- `Receiver` receives two slots (each containing a bit) from the channel and stops

Model2.pml

- Now add a second `Sender` process with exactly the same behavior as the first
- Make `Receiver` expect to receive 0, 1, 0, 1
- Run simulation multiple times with different seeds: what changed?
- Does the verification fail? Why?

If sender1 sends 0 and sender2 sends 0 , but receivers is expecting 0 and 1 intermittedly, it will fail.

Model3.pml

Take `Model2.pml` and modify it such that:

- `Receiver` can initially either receive 0 or 1 first, but it should receive alternating bits thereafter
- Does the verification succeed? Why?

Model4.pml

- Change `Sender` so that it repeats the same behavior indefinitely
- Likewise, `Receiver` repeats the same behavior indefinitely
  - `Receiver` can initially receive 0 or 1 first, but it should receive them in alternating order thereafter
  - If `Receiver` receives a bit out of order, it terminates
- Does the verification succeed? Why?

Model5.pml

Introduce a single backchannel `r2s` of capacity 2 from `Receiver` to `Sender` for error handling

- `r2S` is used to send an `ERR` message when `Receiver` encounters an odd behavior
- If `Receiver` receives a bit out of order (1 after 1 or 0 after 0), it sends an `ERR` message to all Sender processes and terminates
- Any time a `Sender` receives an `ERR` message, it terminates
- If `Sender` terminates, all remaining messages left in any of the channels should be cleared by `Receiver` and `Receiver` terminates
- Use `len(ch)` to read the current length of a channel
- Use `timeout` if necessary

Check that this model doesn't have any deadlocks and passes the verification!

Model6.pml

- Now parameterize the `Sender` process in `Model5.pml` with a start bit value to send, without changing its alternating-bit behavior.
- To invert a bit variable `b`, you can use `b++ or b = 1 – b`
- Can you use `d_step` to reduce the number of states without affecting the behavior?
- The model with these `init` processes should pass all the verifications performed by you and on Vocareum:

```
init {
  run Sender(0);
  run Sender(0);
  run Receiver();
}
```

```
init {
  run Sender(0);
  run Sender(1);
  run Receiver();
}
```

```
init {
  run Sender(1);
  run Sender(0);
  run Receiver();
}
```

```
init {
  run Sender(1);
  run Sender(1);
  run Receiver();
}
```

Model7.pml

Change done => finished

- Introduce two global variables `zeros` and `ones` that count number of 0s and 1s received *in the expected order*
  - Using an assertion, verify that number of 0s and 1s received are always within one count of each other
- Introduced a second id parameter of type byte to Sender to give the Sender instance a unique id (either 0 or 1).
- Introduce global array `finished[]` of size 2 that tracks when a `Sender` process terminates (`finished[0]` is true when first `Sender` process terminates and `finished[1]` is true when second `Sender` process terminates)
  - Using an assertion, verify that both `Sender` processes eventually terminate
  - Use `timeout` if necessary
- Can you remove the repetition in Receiver and make it shorter?
- The model with these `init` processes should not fail the verification with invalid end state or assertion violation:

```
init {
  run Sender(0, 0);
  run Sender(0, 1);
  run Receiver();
}
```

```
init {
  run Sender(0, 0);
  run Sender(1, 1);
  run Receiver();
}
```

```
init {
  run Sender(1, 0);
  run Sender(0, 1);
  run Receiver();
}
```

```
init {
  run Sender(1, 0);
  run Sender(1, 1);
  run Receiver();
}
```

Is the memory or depth limit reached? Why?
What would happen if it could continue with unlimited memory?

Model8.pml

Change done => finished

Review the solution for Model7 to achieve a more concise, and better-parameterized version.

To parameterize: replace the two global variables `zeros` and `ones` with an array `count`, where `count[0]` that counts the number of 0s and `count[1]` counts the number of 1s received *in the expected order*. Have a single Sender proctype with proper parameters: you should be able to instantiate this proctype with the proper arguments to create the two Sender instances.

Also: Add a limit on number of bits sent… *this solution restricts the number of bits sent, and therefore, does not reach the depth or memory limit!*