Distributed Systems

Euan Rochester

May 2018

1 Flow of Messages

1. A key is pressed, synchronising with the plugboard on `keys`.
2. After sending the key inc is sent by the plugboard to the Rotors, and they turn over as needed, forwarding the `inc` message dependant on their current position.
3. The plugboard transforms the input according the f_{plug}.
4. The plugboard sends the message to the first rotor which will transform it according to f_{rotor} and forwards it to the next rotor (synchronisation happens on `mn` where n is the number of the rotor).
5. In the same manner the value undergoing encryption is passed through all the rotors being transformed by their f_{rotor}
6. The last rotor synchronises with the reflector on `ref`, the value is transformed by f_{refl} and passed back to the last rotor once again synchronising on `ref`.
7. Each of the rotors then transforms the value via their $\underline{f_{rotor}}$ function and passes the value to the next outbound rotor until the plugboard is reached again (synchronisation again happens on `mn` where n is the number of the rotor).
8. The plugboard recieves the value on `m3`, transforms it via f_{plug} and passes it back to the keyboard via `keys`.

2 Modifications

As the system is being implemented in Erlang which does not wait for message passing to complete, and inc does not guard l and r in Rotor, it is possible that without an extra set of messages to confirm completion of incrementation, or a restructuring such that inc guards l and r increments will become out of sync with encryption leading to incorrect results. I have also moved incrementation

to be be before key input, as while this is not where icrementation physically happens, it logically happens first. The modification I made can be modeled something like:

$$Keyboard = \overline{inc}.inc_{complete}.\overline{key}\langle x\rangle.lamp(y).Keyboard$$

$$Rotor(26, p, true) = inc.\overline{inc_{complete}}.Rotor(0, p - 26, true) + RotorFunction(p)$$

$$Rotor(26, p, false) = inc.\overline{inc}.inc_{complete}.\overline{inc_{complete}}.Rotor(0, p - 26, false) + RotorFunction(p)$$

$$Rotor(c, p, true) = inc.\overline{inc_{complete}}.Rotor(c + 1, p + 1, true) + RotorFunction(p)$$

$$Rotor(c, p, final) = inc.inc_{complete}.Rotor(c + 1, p + 1, final) + RotorFunction(p)$$

If it is not the final rotor, and is turning over it must increment the next rotor. If it is the final rotor it must *always* signal increment complete. Otherwise it simply increments, and signals complete.

Obviously this would also need suitable renaming done to connect each $inc_{complete}$ back to the previous rotor, and also from the first rotor to the keyboard.

3 Testing

Testing has primarily been done in a property validation style, using the free "mini" version of Quviq quickcheck for erlang. An example of a property that is tested is: feeding a letter through a rotor, then back through that rotor in reverse should yield the original letter. Tests are in `property_tests.erl`