COM3190/COM6116 Programming Assignment
Erl-nigma – A simulated Enigma machine

Ramsay Taylor
Department of Computer Science
The University of Sheffield

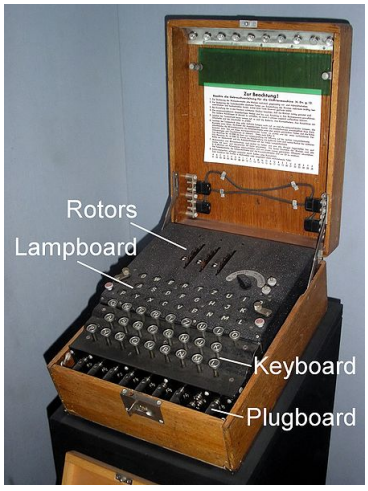
DEADLINE: 3.00pm, Wednesday, Week 12

Guidelines

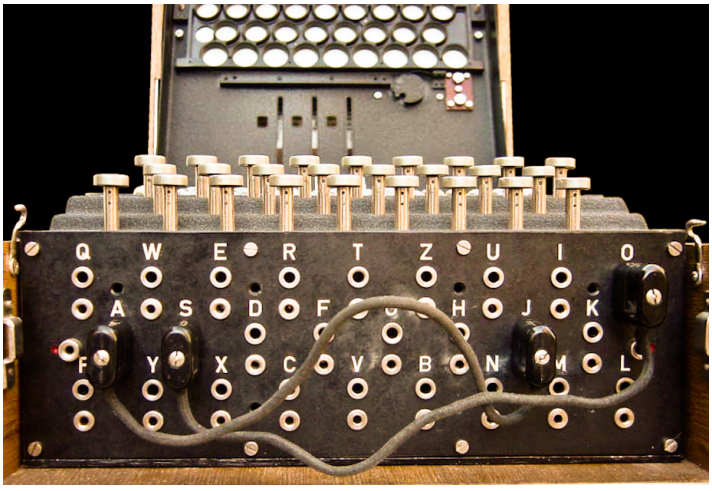
This assignment contributes 30% towards your overall module mark, and requires you to submit TWO files: (a) a single PDF document answering Part 1 below; (b) a single Erlang program file containing your solutions to Part 2. You should submit your solutions via MOLE no later than the deadline specified above. Standard penalties will apply for late submissions.

Background

The Enigma machine was invented in Germany in 1918, and uses various mechanical and electrical components to encrypt and decrypt messages, one letter at a time. Originally developed for commercial use, it was soon adapted for military and diplomatic purposes (Figure 1a) [5].



(a) Enigma



(b) Plugboard

Figure 1: (a) A Military Enigma Machine [6]. (b) Plugboard [3]. The wires shown here swap A with J, and S with O.

Figure 2: Rotors [2].

Components

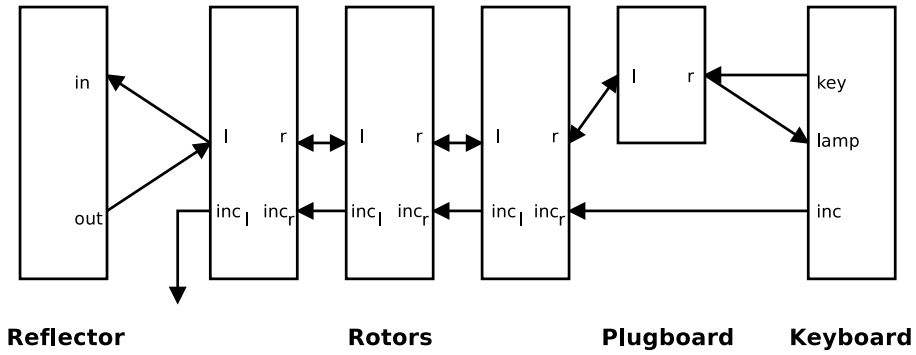


Figure 3: An outline of the Enigma process

Figure 3 shows the basic components of an Enigma machine and their interconnections:

Keyboard
: The machine has a *keyboard*, on which users type their plaintext messages.

Plugboard
: The *plugboard* is used to set up a basic substitution cipher. In Figure 1b, for example, a wire has been used to join the sockets labeled A and J. This means that every time the user types the letter A, the keyboard actually produces the letter J (and vice versa). Similarly, because there is a wire joining sockets S and O, typing the letter O actually produces the letter S (and vice versa).

Rotor
: The machine is equipped with several rotors, which again perform simple letter-for-letter substitutions. Each rotor is connected to the next by electrical contacts, and the result of sending a letter through all of the rotors, one after the other, depends on their positions relative to one another (Figure 2). The power of the Enigma machine comes from the simple tactic of changing the relative positions of the rotors after each letter has been encrypted. Just because E generated the output Z last time (say), doesn't mean it will generate the output Z this time, because the rotors will have rotated relative to one another.

Reflector, *Lampboard*
: Once the current letter has been transmitted through the plugboard and rotors, it is passed through a *reflector* (this again connects letters together in pairs). Having passed through the reflector, the letter undergoes further encryption by being passed back through the rotors and plugboard. On reaching the keyboard, the built-in *lampboard* shows the user which letter should be used to encrypt the letter they just typed.

Modelling the details of the encryption process in CCS would require modelling each of the individual letter connections on each rotor as a separate channel, which would make for an unwieldy model. Modelling the system in π-calculus would allow you to represent the changing rotor connections more economically, but it would still require 26 channels per rotor, and an explicit definition of the internal connections of each.

Instead, a more abstract model can be constructed that includes the components and their connections, but represents the actual operations over the characters by declaring the functions used to manipulate the data as it flows across the channels. One such model is shown in Figure 4.

$$
\begin{array}{lcl}
\mathit{Reflector} & = & \mathit{in}(x).\overline{\mathit{out}}\langle f_{\mathit{refl}}(x)\rangle.\mathit{Reflector} \\
\\
\mathit{Keyboard} & = & \overline{\mathit{key}}\langle x\rangle.\overline{\mathit{inc}}.\mathit{lamp}(y).\mathit{Keyboard} \\
\\
\mathit{Plugboard} & = & r(x).\overline{l}\langle f_{\mathit{plug}}(x)\rangle.\mathit{Plugboard} \\
 & + & l(x).\overline{r}\langle f_{\mathit{plug}}(x)\rangle.\mathit{Plugboard} \\
\\
\mathit{Rotor}(26,p) & = & \mathit{inc}_r.\overline{\mathit{inc}_l}.\mathit{Rotor}(0,p-26) + \mathit{RotorFunction}(p) \\
\mathit{Rotor}(c,p) & = & \mathit{inc}_r.\mathit{Rotor}(c+1,p+1) + \mathit{RotorFunction}(p) \\
\mathit{RotorFunction}(p) & = & l(x).\overline{r}\langle f_{\mathit{rotor}}(p,x)\rangle.\mathit{RotorFunction}(p) \\
 & + & r(x).\overline{l}\langle \overline{f_{\mathit{rotor}}}(p,x)\rangle.\mathit{RotorFunction}(p) \\
\\
\mathit{Enigma} & = & \mathit{Reflector}[\mathit{ref}/\mathit{in}, \mathit{ref}/\mathit{out}] \\
 & | & \mathit{Rotor}(c_3,p_3)[\mathit{ref}/l, m1/r, i3/\mathit{inc}_r] \\
 & | & \mathit{Rotor}(c_2,p_2)[m1/l, m2/r, i3/\mathit{inc}_l, i2/\mathit{inc}_r] \\
 & | & \mathit{Rotor}(c_1,p_1)[m2/l, m3/r, i2/\mathit{inc}_l, i1/\mathit{inc}_r] \\
 & | & \mathit{Plugboard}[m3/l, \mathit{keys}/r] \\
 & | & \mathit{Keyboard}[\mathit{keys}/\mathit{key}, \mathit{keys}/\mathit{lamp}, i1/\mathit{inc}]
\end{array}
$$

Figure 4: An abstractly parametrised CCS/π-calculus model of the Enigma system components.

The *Reflector* process simply takes an input on channel *in* and reflects it out on channel *out* having modified it with the reflection function f_{refl}. The *Keyboard* process outputs a keypress on channel *key* (the choice of key is not specified), outputs an increment signal on channel *inc* to advance the rotors, and the waits for a signal on channel *lamp* to represent the result from the encryption process being received on the lamp board. The plugboard simply passes messages from l to r or r to l via the plugboard function f_{plug}. Since the real plugboard is implemented with wires tying letters together, f_{plug} is bijective — so, if a is translated to x then x is translated to a — and so there doesn't need to be an inverse function.[1]

The rotor encryption function is modelled in *RotorFunction*, which is parameterised with a position, p. It will accept a character on the left channel, l, and output on channel r with the character modified by the rotor function f_{rotor}, which takes both the current rotor position and the character as arguments and models the current passing through the rotor wiring from right to left. Alternatively, *RotorFunction* can accept a character on the right channel, r, and output on l having passed the character through the inverse encryption function $\overline{f_{\mathit{rotor}}}$, which represents the current passing through the rotor wiring from left to right. Different internal rotor wirings can be represented with different functions.[2]

The operation of an individual rotor is a combination of its encryption function, specified in *RotorFunction*, and its physical rotation. The *Rotor* process is parametrised by both its current position, p, and by counting the number of increments (c) that have been made so far. Once a rotor has been incremented 26 times it will pass on an increment signal to its left-hand neighbour,

[1] This detail was essential to the process of cracking Enigma using the *bombe* computers at Bletchley Park [4].

[2] Traditionally the Enigma machine was supplied with several different rotors, each labeled with a Roman numeral (I, II, III, etc.). Part of the encryption key for an Enigma cypher is the rotor selection and ordering.

and return to its original position (i.e. it will have completed a full rotation).[3]

The complete *Enigma* process considered in this assignment combines the reflector, three rotors, and the keyboard into a parallel composition. The channels are connected by renaming, using the internal labels shown in Figure 5.

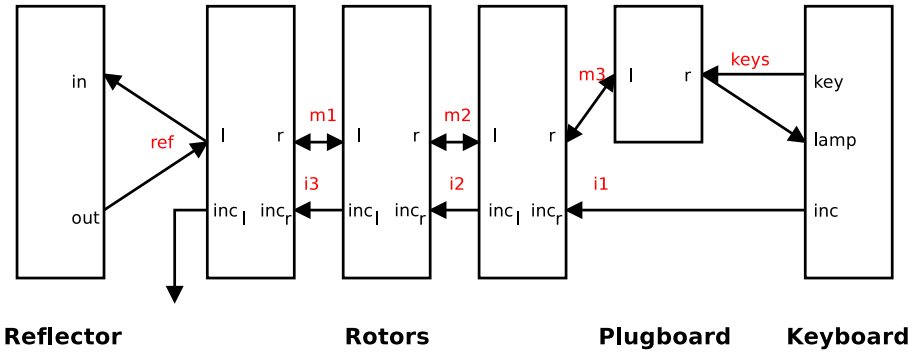


Figure 5: The internal channel names

Part 1 (40%)

In the real Enigma machine, pressing a key both turned on an electric current on the appropriate channel and advanced the first rotor. Since the machine was operated by a human and keys were held for a second or so, the real world process of using an enigma would allow the rotors to settle in their new position *after being rotated* before a lamp was read.

The *Keyboard* process in Figure 4 sends a key press signal and then an increment signal, and then waits for a lamp signal. Does this mean that the model correctly implements the physical system? Is the keypress character passed through the rotors in their newly-incremented position before being returned to the lamp? Do the *Rotor* processes interoperate safely, or is there a potential for deadlock in the model (which clearly isn't present in the physical system)?

(a) Describe the flow of messages through the concurrent system, the possible synchronisations, and the possible sequences of messages. Identify any problems. **[15%]**

(b) Modify the model to make it a more accurate representation of the real Enigma system, and explain briefly how your version overcomes the problems you identified in (a). **[25%]**

Part 2 (60%)

Build an Erlang system to implement the Enigma process. You should submit a single Erlang file that implements a module named `enigma`. Internally, this should implement separate processes for at least the six components shown in Figure 3 — the *Keyboard*, the *Plugboard*, the three *Rotor*s (which can be three process instances of the same function if you like), and the *Reflector*. You can create extra processes for managing the Enigma and its communications with the outside world if you think it appropriate.

[3]The actual details are slightly more complicated than this. The rotors were equipped with notches that determined at what point the increment signal would be passed to the left-hand neighbour.

The following functions should be exported from your module:

- `setup/5` – which accepts 5 arguments: a reflector name, a triple of rotor names, a triple of ring-settings, a list of plugboard pairs, and an initial setting, and returns a PID of an enigma machine; e.g.,

```
Enigma  =  setup("B", {"III","I","II"}, {12,6,18},
                                [{$A,$E},{$F,$J},{$P,$R}], {$D,$F,$R})
```

- `crypt/2` – which accepts 2 arguments: a PID of an Enigma machine (produced by `setup/5`), and a string of text to be passed through the machine. It should handle all communications with the machine and return the string of responses from the machine.

- `kill/1` – which accepts the PID of an Enigma machine (produced by `setup/5`) and shuts it down, terminating all the internal processes.

Note that in Erlang the dollar prefix allows you to use ASCII character values (e.g. `$A` is 65). When encrypting and decrypting, the machine should treat the characters "a" and "A" identically. Converting all input to uppercase (inside `crypt/2`) and returning uppercase output is a legitimate solution. You can either pass other characters through unchanged, or strip out other characters entirely. For historical realism you could even group everything into blocks of 5 characters! You won't lose marks for any of these approaches, so long as your choice is sensible, and you explain in the comments what you have chosen to do.

To save you typing, I have written an Erlang header file that defines the letter mappings for the rotors and reflectors (see [7] for the download link). They are defined as lists of pairs that represent the f_{rotor} functions. There are more succinct representations, so feel free to convert these if it suits you better. Your module should be able to support rotors `I` through `V`, but the higher numbered rotors increment twice, and this would require your rotor code to be more complicated. The module should support all reflectors, and it should allow arbitrary plugboard and ring settings.

The majority of the marks for Part 2 will be for the implementation [**30%**] of the processes and their interactions, so you should try to produce generalised processes that can use this data. The `lists:keyfind/3` function [1] may be helpful! We will also be looking for sensible commenting [**10%**] of your code, which should include brief details of how you tested it [**20%**].

References

[1] Ericsson AB. Erlang Lists Module API Documentation. `http://www.erlang.org/doc/man/lists.html` [Accessed 20th Oct. 2015].

[2] Ted Coles. Enigma rotors and spindle showing contacts rachet and notch. Photo: `https://commons.wikimedia.org/wiki/File:Enigma_rotors_and_spindle_showing_contacts_rachet_and_notch.jpg` [Accessed 1 March 2016], 2011.

[3] Bob Lord. Enigma plugboard. Photo: `http://commons.wikimedia.org/wiki/File:Enigma-plugboard.jpg` [Accessed 1 March 2016], 2005.

[4] Tony Sale. Alan Turing, the Enigma and the Bombe. `http://www.codesandciphers.org.uk/virtualbp/tbombe/tbombe.htm` [Accessed 20th Oct. 2015].

[5] Tony Sale. The Components of the Enigma Machine. `http://www.codesandciphers.org.uk/enigma/enigma2.htm` [Accessed 20th Oct. 2015].

[6] Karsten Sperling. Enigma Machine at the Imperial War Museum, London. Photo: `http://commons.wikimedia.org/wiki/File:EnigmaMachineLabeled.jpg` [Accessed 1 March 2016], 2005.

[7] Ramsay Taylor. Rotor and Reflector Header File. http://staffwww.dcs.shef.ac.uk/people/R.Taylor/COM3190/Enigma.hrl [Accessed 20th Oct. 2015].