1 CCS Model

1.1 Problems of Default CCS Model Definiton

1. **Livelock scenario:** The would happen when the system has at least one O and OH spawned. O can evolve into OH, and OH can evolve into O, this would cause the system to synchronise O and OH continuously.
2. **Deadlock scenario:** By working through the CCS processes manually, the system will reach a state where it would require to synchronise on either an o or h in order to evolve two of the processes. However, in order for two processes to synchronise on o, the system must at least synchronise on h before. This introduces a deadlock because o and h have mutual dependency.

1.2 Improved CCS Model

The objective is to eliminate the problems stated in **Section 1.1**. The accepted intermediate states are: $CH_3, CH_4, HCO, CH_3O, CH_2OH$. The rule highlighted in bold is added to ensure that the system can simulate the equation (1).

$$
\begin{aligned}
CH_3OH &= \overline{c}.(H_2 \mid H \mid OH) + \overline{h}.CH_2OH + \overline{oh}.CH_3 \\
CH_2OH &= \overline{c}.(H \mid H \mid OH) + \overline{h}.(COH \mid H) + \overline{oh}.(C \mid H_2) \\
COH &= \overline{c}.OH + \overline{oh}.C \\
\\
CH_3 &= \overline{c}.(H_2 \mid H) + \overline{h}.(C \mid H_2) \\
\\
H_2 &= \boldsymbol{o.H_2O} + \overline{h}.H \\
H &= oh.H_2O + \overline{h}.0 \\
\\
O_2 &= c.CO_2 + \overline{o}.O \\
O &= h.OH + \overline{o}.0 \\
OH &= h.H_2O + \overline{o}.H \\
C &= o.CO \\
CO &= o.CO_2
\end{aligned}
$$

2 Erlang Implementation

2.1 Design

It is known that a complete combustion of methanol in oxygen will form carbon dioxide and water, which is described as the balanced equation below:

$$2\,\mathrm{CH_3OH} + 3\,\mathrm{O_2} \longrightarrow 2\,\mathrm{CO_2} + 4\,\mathrm{H_2O} \qquad (1)$$

The primary objective of the Erlang implementation is to simulate the equation (1). The system needs to satisfy the requirements below:

1. When the ratio of CH_3OH to O_2 is 2 to 3, the system must undergo complete combustion and produce CO_2 and H_2O of ratio 2 to 4.
2. When the ratio of CH_3OH to O_2 is **not** 2 to 3, the system will undergo incomplete combustion and produce incomplete amount of CO_2 and H_2O, along with some intermediate products.
3. At the end of the reaction, the system must be able to detect whether it is a complete or incomplete combustion.

2.2 Program Flow Diagram

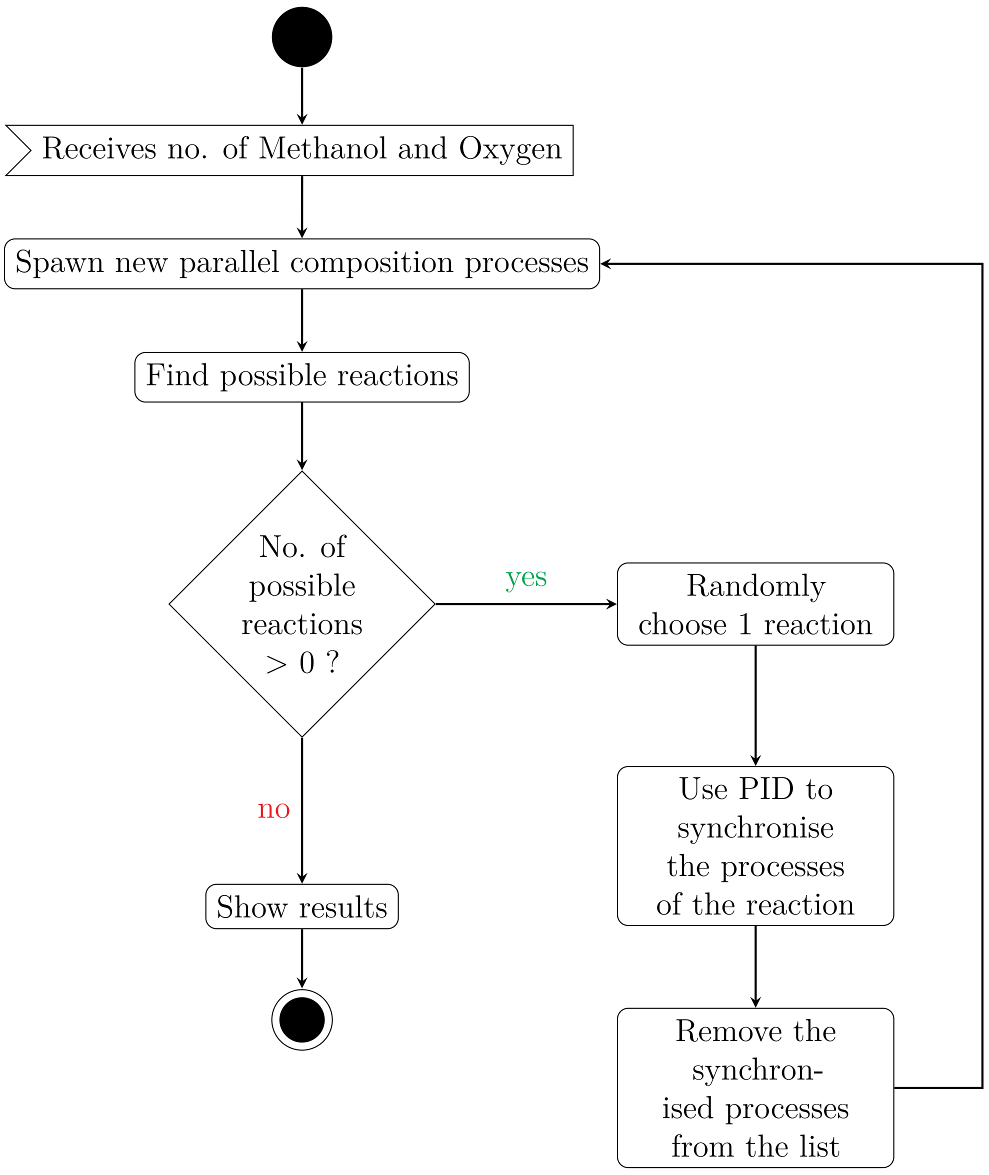
Receives no. of Methanol and Oxygen
Spawn new parallel composition processes
Find possible reactions
No. of possible reactions > 0 ?
yes
Randomly choose 1 reaction
Use PID to synchronise the processes of the reaction
Remove the synchron-ised processes from the list
no
Show results

Figure 1: Flow diagram of the Erlang implementation

2.3 Implementation Details

This section will discuss design concepts and implementation details of the system to provide explanation for the program flow in **Figure 1**.

2.3.1 Rule set

In the Erlang implementation, process synchronisation is done by sending and receiving messages through process identifier (PID). The system itself does not know which processes could synchronise with each other. Therefore, a rule set must be integrated to allow the system to check for possible reactions.

```
Rules = #{
  ch3oh => [{snd, c}, {snd, h}, {snd, oh}],
  ch2oh => [{snd, c}, {snd, h}, {snd, oh}],
  coh => [{snd, c}, {snd, oh}],
  ...
  o2 => [{rcv, c}, {snd, o}],
  ...
},
```

Listing 1: A rule set definition in Erlang maps [2]

A partial rule set is shown in **Listing 1**. The atom `snd` and `rcv` represent 'send' and 'receive' respectively. For two processes to synchronise, one must be able to '`snd`', and the other must be able to '`rcv`' the same atom. The system will use this mechanism to find possible reactions. For instance, `ch3oh`, `ch2oh`, `coh` can '`snd`' a `c`, and `o2` can '`rcv`' a `c`. Hence, the system will know that `ch3oh`, `ch2oh`, and `coh` can all synchronise with `o2`.

2.3.2 Reactants List

Moreover, the system also needs to have a variable to store the reactants (processes spawned) left. The reactants list is used by the system to find possible reactions, and to determine whether the reaction has ended. The reactants list store a list of reactants in the form `{atom, PID}`, where `atom` represents the name of the reactant, and `PID` represents the process identifier of the reactant.

```
[{ch3oh,<0.98.0>},{h,<0.106.0>},{o2,<0.99.0>},{oh,<0.107.0>},...]
```

Listing 2: An example of reactants list.

2.3.3 Initialising the Start Process

Since the system should only terminate when there are no possible reactions left, and synchronising two processes would generally involve spawning processes as well, a start process is implemented to handle the main logic of the system based on **Figure 1**. The start process is initially registered when the user has provide the number of Methanol and Oxygen molecules, and it is kept alive until the end of the reaction. It is a recursive function, the new processes spawned after a synchronisation are sent back, and then it will start to find possible reactions again.

```
-spec start_reaction([{atom(), pid()}]) -> none().
% @doc Main logic, start the combustion of methanol
% @param Reactants The list of reactants

start_reaction(Reactants) ->
  Reaction = find_reaction(Reactants),
  case Reaction of
    % No possible reactions left, end and show results
    [] -> result ! {finished, Reactants};

    % Randomly chosen reaction returned
    {{R1, _}, {R2, _}, _} ->
      perform_reaction(Reaction),

      % Remove the two reactants reacted after the reaction
      NewReactants = remove_reactants(R1, R2, Reactants),

      % Wait until the reaction has finished, then repeat again
      receive
        NewProcesses ->
          start_reaction(lists:merge(NewProcesses, NewReactants))
      end
  end.
```

Listing 3: A function that handles the main logic of the system

2.3.4 Finding Possible Reactions

With the rule set and reactants list defined, the system can now find possible reactions in the reactants list. Given a reactants list, the system will process the list recursively, searching for reactants that can synchronise with each other. All possible reactions will be stored in a list in the format shown in **Listing 4**. In order to simulate the 'randomness' behaviour of chemistry and the 'choice' behaviour of CCS, the reaction to perform will be randomly selected from the list of possible reactions. If no possible reactions are found, then this indicates that the reaction has ended, and the system will show the results.

```
{{{ch3oh, <0.98.0>}, snd}, {{o2, <0.99.0>}, rcv}, c}
```

Listing 4: An example of a possible reaction

2.3.5 Synchronising Processes

After finding a reaction to perform, the system will then simulate the 'synchronisation' behaviour. According to **Listing 4**, all the required information to perform the synchronisation has been recorded. Using **Listing 4** as an example, the system can extract the following information:

1. `ch3oh` can synchronise with `o2` on `c`
2. The PID of `ch3oh`

3. The PID of o2
4. `ch3oh` is the sender
5. `o2` is the receiver

Since `ch3oh` is the sender, it should send `c` to `o2` and evolve. However, in Erlang implementation, the `ch3oh` process itself does not know its identity. Therefore, the system must first send a message to `ch3oh`, telling it to send `c` to the PID of `o2` in order to synchronise the processes. The function to initiate the synchronisation process is shown in **Listing 5**.

```
-spec perform_reaction(reaction()) -> none().
% @doc Perform a reaction
% @param A reaction

perform_reaction({{{R1, R1_Pid}, R1_Comm}, {{R2, R2_Pid}, R2_Comm}, Action}) ->
  case {R1_Comm, R2_Comm} of
    {snd, rcv} ->
      % Reactant 1 sends Action to Reactant 2
      R1_Pid ! {snd, Action, R2_Pid},
    {rcv, snd} ->
      % Reactant 2 sends Action to Reactant 1
      R2_Pid ! {snd, Action, R1_Pid},
  end.
```

Listing 5: A function to perform the reaction

2.3.6 Evolving and Spawning New Processes

```
1  ch3oh() ->
2    receive
3      {snd, c, To} ->
4        To ! {rcv, c, ch3oh, self()},
5        receive
6          {ok, R} -> start ! spawn_reactants([h2, h, oh] ++ R)
7        end;
8      ...
9    end.
10
11 o2() ->
12   receive
13     {rcv, c, From, Pid} ->
14       co2(),
15       Pid ! {ok, []};
16     ...
17   end.
```

Listing 6: Partial code listing of `ch3oh` and `o2`

In Erlang, message sending is asynchronous [1]. For example, in **Listing 6**, after `ch3oh` has sent a `c` to `o2`, it will then wait for a response from `o2` (line 6). The response message will contain `ok`, along with a list of new processes to spawn (line 15, in this case the list is empty because O_2 has become CO_2). As mentioned in **Section 2.3.3**, new processes spawned after the synchronisation will be sent back to the start process (line 6). After that, the synchronisation is considered complete and the system can proceed to find new possible reaction.

2.3.7 Traces

In order to prove that the system implemented matches the CCS model, one possible trace of the system is shown in **Listing 7**. It has been shown that the processes synchronised are removed from the reactants list, and new processes spawned are added into the reactants list. Moreover, **Listing 7** has also shown that the system has fulfilled the requirements set in **Section 2.1**.

```
> chemistry:react(4,2).

Reactants left: [ch3oh,ch3oh,ch3oh,ch3oh,o2,o2]
ch3oh --c--> o2
o2 <--c-- ch3oh

Reactants left: [ch3oh,ch3oh,ch3oh,h2,h,o2,oh]
ch3oh --h--> oh
oh <--h-- ch3oh

Reactants left: [ch2oh,ch3oh,ch3oh,h2,h,o2]
ch2oh --c--> o2
o2 <--c-- ch2oh

Reactants left: [ch3oh,ch3oh,h,h,h2,h,oh]
ch3oh --h--> oh
oh <--h-- ch3oh

Reactants left: [ch2oh,ch3oh,h,h,h2,h]
ch2oh --oh--> h
h <--oh-- ch2oh

Reactants left: [c,ch3oh,h,h,h2,h2]
ch3oh --oh--> h
h <--oh-- ch3oh

Reactants left: [c,ch3,h,h2,h2]
Final products: 2CO2 + 4H2O + 2H2 + 1C + 1CH3 + 1H
Status       : Run out of O2
-------------------------------------------------------------------
> chemistry:react(2,3).

Reactants left: [ch3oh,ch3oh,o2,o2,o2]
ch3oh --c--> o2
o2 <--c-- ch3oh

Reactants left: [ch3oh,h2,h,o2,o2,oh]
```

```
ch3oh --h--> oh
oh <--h-- ch3oh

Reactants left: [ch2oh,h2,h,o2,o2]
ch2oh --oh--> h
h <--oh-- ch2oh

Reactants left: [c,h2,h2,o2,o2]
o2 --o--> c
c <--o-- o2

Reactants left: [h2,h2,o,co,o2]
o --o--> h2
h2 <--o-- o

Reactants left: [h2,co,o2]
o2 --o--> h2
h2 <--o-- o2

Reactants left: [co,o]
o --o--> co
co <--o-- o

Reactants left: []
Final products: 2CO2 + 4H2O
Status      : Complete combustion
```

Listing 7: Traces of the system

2.4 Testing

EUnit testing has been done to cover the functionalities of some helper functions.

```
> chemistry_test:test().
  All 42 tests passed.
ok
```

Listing 8: EUnit testing

2.5 Conclusion

1. The system implemented is functional, and is able to meet the requirements set.
2. The system is able to simulate the 'randomness' behaviour of chemistry reaction, as well as the CCS choices behaviour by randomly selecting one reaction from a list of possible reactions.
3. Erlang is able to simulate the CCS synchronisation by message sending and receiving between processes.
4. Different definitions of CCS model may produce different results on the same input.

References

[1] *Erlang reference manual.* [Online]. Available: `http://erlang.org/doc/reference_manual/processes.html#message-sending` (visited on 07/05/2019).

[2] *Erlang stdlib: Maps.* [Online]. Available: `http://erlang.org/doc/man/maps.html` (visited on 07/05/2019).

Appendix

Default CCS Definition

$$\begin{aligned}
CH_3OH &= \overline{c}.H_3OH + \overline{h}.CH_2OH + \overline{oh}.CH_3 \\
CH_2OH &= \overline{c}.H_2OH + \overline{h}.CHOH + \overline{oh}.CH_2 \\
CHOH &= \overline{c}.HOH + \overline{h}.COH + \overline{oh}.CH \\
COH &= \overline{c}.OH + \overline{oh}.C \\
\\
CH_3 &= \overline{c}.H_3 + \overline{h}.CH_2 \\
CH_2 &= \overline{c}.H_2 + \overline{h}.CH \\
CH &= \overline{c}.H + \overline{h}.C \\
\\
H_3OH &= \overline{h}.H_2OH + \overline{oh}.H_3 \\
H_2OH &= \overline{h}.HOH + \overline{oh}.H_2 \\
HOH &= \overline{h}.OH + \overline{oh}.H \\
\\
H_3 &= \overline{h}.H_2 \\
H_2 &= \overline{h}.H \\
H &= oh.H_2O + \overline{h}.0 \\
\\
O_2 &= c.CO_2 + \overline{o}.O \\
O &= h.OH + \overline{o}.0 \\
OH &= h.H_2O + \overline{o}.H + \overline{h}.O \\
C &= o.CO \\
CO &= o.CO_2
\end{aligned}$$