**Assignment 6**

Submitted By: **Siri Anil**

**1B. Explain how you abstracted your implementation to UPPAAL models?**

The implementation of the Dining Philosophers model is as described below.

The first and the initial state is Thinking. The thinking state has an invariant of x<=10 which means that the philosopher cannot stay in this state for more than 10s. From the thinking state, the next state is the Hungry State. Once the philosopher enters the hungry state, since it is committed, he is forced to go to the next state which is takeforks. In the takeforks state, he checkes if both the forks are available. If they are available, then he picks up the forks and begins eating. If it is not, he goes into a waiting state which is called ‘dintgetforks’.

There is an invariant on the eating state, which is x<=2. Hence the philosopher cannot eat for more than 2s. After this, he comes out of eating state and then checks for the neighbors. If either of the neighbors are blocked, i.e,. if they are in didntgetforks state, it unblocks them. This is done by using a channel. After unblocking potentially blocked neighbors, he then enters the thinking state.

Each philosopher can come out of the waiting state and then check if the forks are available only when the neighbor who had blocked it sends it a sync message to unblock it.

The various data variables used are as follows:

* Fork\_1 to fork\_5 – to represent the five forks.
* Id to represent the number of the philosopher.
* Unblock[5] – channel to send a message to unblock.
* Blocked[5] – variable to check it the philosopher is in blocked state.

The queries are as shown below.

1. To show that system will not deadlock – A[] not deadlock. Property is satisfied.
2. To show philosophers won’t enter critical region at the same time – Mutual exclusion

A[] not (P1.Eating and P2.Eating) - Property is satisfied.

Ensures that P1 and P2 are not eating at the same time

A[] not (P2.Eating and P3.Eating) - Property is satisfied.

Ensures that P3 and P2 are not eating at the same time

A[] not (P3.Eating and P4.Eating) - Property is satisfied.

Ensures that P3 and P4 are not eating at the same time

A[] not (P4.Eating and P5.Eating) - Property is satisfied.

Ensures that P4 and P5 are not eating at the same time

A[] not (P5.Eating and P1.Eating) - Property is satisfied.

Ensures that P5 and P1 are not eating at the same time

A[] P5.Eating imply !P4.Eating && !P1.Eating

Ensures that when P5 is eating P4 and P1 cannot be eating

A[] P1.Eating imply !P5.Eating && !P2.Eating

Ensures that when P1 is eating P5 and P2 cannot be eating

A[] P4.Eating imply !P3.Eating && !P5.Eating

Ensures that when P4 is eating P3 and P5 cannot be eating

A[] P2.Eating imply !P3.Eating && !P1.Eating

Ensures that when P2 is eating P3 and P1 cannot be eating

A[] P3.Eating imply !P4.Eating && !P2.Eating

Ensures that when P3 is eating P4 and P2 cannot be eating

1. To show liveness property –

P1.Hungry 🡪 P1.Eating - Property is satisfied.

Whenever P1 is hungry, P1 gets to eat

P2.Hungry 🡪 P2.Eating - Property is satisfied.

Whenever P2 is hungry, P2 gets to eat

P3.Hungry 🡪 P3.Eating - Property is satisfied.

Whenever P3 is hungry, P3 gets to eat

P4.Hungry 🡪 P4.Eating - Property is satisfied.

Whenever P4 is hungry, P4 gets to eat

P5.Hungry 🡪 P5.Eating - Property is satisfied.

Whenever P5 is hungry, P5 gets to eat