Formal Verification

Paul Wild

Wednesday 6th November, 2024

Message Channels

- Channels are used to model the exchange of data between processes.
- A channel is a fifo queue holding data of a specified type.
- They are declared as follows:

```
chan mychannel = [4] of {mtype,int,bool};
```

Message Channels

- Channels are used to model the exchange of data between processes.
- A channel is a fifo queue holding data of a specified type.
- They are declared as follows:

```
chan mychannel = [4] of {mtype,int,bool};
```

- The number gives the size of the queue (here 4), the part following `of` is the type of data (here triples `{mtype,int,bool}`).

Message Channels

- Channels are used to model the exchange of data between processes.
- A channel is a fifo queue holding data of a specified type.
- They are declared as follows:

```
chan mychannel = [4] of {mtype,int,bool};
```

- The number gives the size of the queue (here 4), the part following `of` is the type of data (here triples `{mtype,int,bool}`).
- Messages can be sent using `mychannel!blue,42,true` and received using `mychannel?red,n,b`.

Message Channels

- Channels are used to model the exchange of data between processes.
- A channel is a fifo queue holding data of a specified type.
- They are declared as follows:

```
chan mychannel = [4] of {mtype,int,bool};
```

- The number gives the size of the queue (here 4), the part following `of` is the type of data (here triples `{mtype,int,bool}`).
- Messages can be sent using `mychannel!blue,42,true` and received using `mychannel?red,n,b`.
- Sending blocks the current process if the channel is full and receiving blocks the current process if the channel is empty or the first element does not match the pattern after `?` (in the example above the message type must be `red`).

Message Channels

- Channels are used to model the exchange of data between processes.
- A channel is a fifo queue holding data of a specified type.
- They are declared as follows:

```
chan mychannel = [4] of {mtype,int,bool};
```

- The number gives the size of the queue (here 4), the part following `of` is the type of data (here triples `{mtype,int,bool}`).
- Messages can be sent using `mychannel!blue,42,true` and received using `mychannel?red,n,b`.
- Sending blocks the current process if the channel is full and receiving blocks the current process if the channel is empty or the first element does not match the pattern after `?` (in the example above the message type must be `red`).
- Rendezvous points between processes can be modelled using a channel of size 0.

Dining Philosophers

Setting

- n philosophers are sitting around a circular dining table and there are n forks, one between each pair of adjacent philosophers.
- Whenever a philosopher is hungry, they first grab the fork to their left, then the fork to their right, and then they eat.
- When they are done they release both forks.

Dining Philosophers

Setting

- n philosophers are sitting around a circular dining table and there are n forks, one between each pair of adjacent philosophers.
- Whenever a philosopher is hungry, they first grab the fork to their left, then the fork to their right, and then they eat.
- When they are done they release both forks.

Modelling in Promela

We will now try to model this in Promela.

- The philosophers will be modelled as processes (`proctype`) which we create from the `init` process using the `run` command.
- The forks will be modelled as channels (`chan`).

What are some properties of this system that we may want to check?

Model checking strategies

- If we run this model with Spin, it reports that a deadlock can occur.
- Investigation of the trace shows that it is quite long. How can we shorten it?

Model checking strategies

- If we run this model with Spin, it reports that a deadlock can occur.
- Investigation of the trace shows that it is quite long. How can we shorten it?
- Method 1: Using iterative deepening depth first search:

```
gcc -DREACH pan.c -o pan
./pan -i
```

Model checking strategies

- If we run this model with Spin, it reports that a deadlock can occur.
- Investigation of the trace shows that it is quite long. How can we shorten it?
- Method 1: Using iterative deepening depth first search:

```
gcc -DREACH pan.c -o pan
./pan -i
```

- Method 2: Using breadth first search:

```
gcc -DBFS pan.c -o pan
./pan
```

Model checking strategies

- If we run this model with Spin, it reports that a deadlock can occur.
- Investigation of the trace shows that it is quite long. How can we shorten it?
- Method 1: Using iterative deepening depth first search:

```
gcc -DREACH pan.c -o pan
./pan -i
```

- Method 2: Using breadth first search:

```
gcc -DBFS pan.c -o pan
./pan
```

- Verify that the paths generated in this way are indeed as short as possible.

Dining Philosophers

Deadlocks

- Model checking showed us that a deadlock can occur when all philosophers simultaneously grab the fork to their left.
- What are some possible changes to our model that ensure that this is no longer possible?

Dining Philosophers

Deadlocks

- Model checking showed us that a deadlock can occur when all philosophers simultaneously grab the fork to their left.
- What are some possible changes to our model that ensure that this is no longer possible?

Approaches

1. Use atomic sequences. How?

Dining Philosophers

Deadlocks

- Model checking showed us that a deadlock can occur when all philosophers simultaneously grab the fork to their left.
- What are some possible changes to our model that ensure that this is no longer possible?

Approaches

1. Use atomic sequences. How?
2. Relax the requirement that everybody needs to first grab the fork to their left. How?

Dining Philosophers

Deadlocks

- Model checking showed us that a deadlock can occur when all philosophers simultaneously grab the fork to their left.
- What are some possible changes to our model that ensure that this is no longer possible?

Approaches

1. Use atomic sequences. How?
2. Relax the requirement that everybody needs to first grab the fork to their left. How?
3. Allow the possibility of releasing the left fork before grabbing the right. How?

Dining Philosophers

Deadlocks

- Model checking showed us that a deadlock can occur when all philosophers simultaneously grab the fork to their left.
- What are some possible changes to our model that ensure that this is no longer possible?

Approaches

1. Use atomic sequences. How?
2. Relax the requirement that everybody needs to first grab the fork to their left. How?
3. Allow the possibility of releasing the left fork before grabbing the right. How?
4. Introduce a waiter that needs to be asked for permission first. How?