

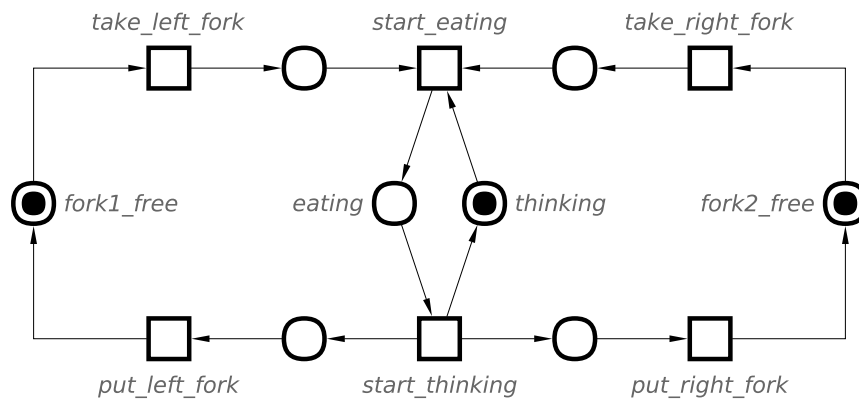
## Dining philosophers problem

In this tutorial we will use the famous [Dining philosophers problem](http://en.wikipedia.org/wiki/Dining_philosophers_problem)

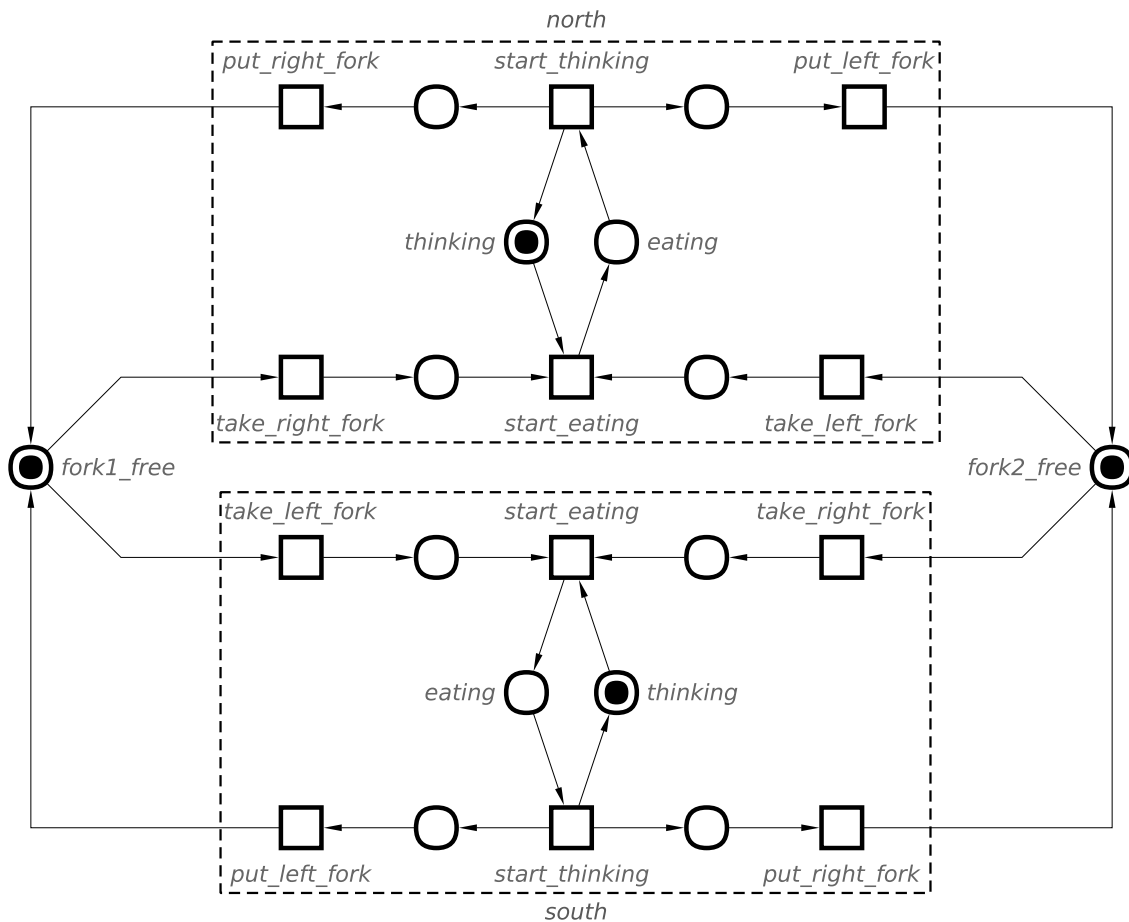
[[http://en.wikipedia.org/wiki/Dining\\_philosophers\\_problem](http://en.wikipedia.org/wiki/Dining_philosophers_problem)] to illustrate the power and convenience of the Petri nets. The problem states that there are five philosophers sitting around a table. Each philosopher either thinks or eats. There are five forks on the table shared among philosophers in such a way that each fork can be either used by a philosopher on its left or on its right. Before eating a philosopher tries to take both forks in some order. If a fork is already taken by a neighbouring philosopher, he will be waiting until the fork becomes available again. After eating, a philosopher returns to thinking while putting back both of his forks.

### Modelling




Petri nets is an ideal tool for formally modelling the dining philosophers problem. The behaviour of a single philosopher can be modelled with `take_left_fork`, `put_left_fork`, `take_right_fork`, `put_right_fork`, `start_eating` and `start_thinking` transitions. The state of the philosopher is captured with a token in either the eating or thinking place and the availability of forks is determined by the presence of a token in `fork1_free` and `fork2_free` places respectively (the corresponding fork is on the table), as shown in the following Petri net.



For simplicity let us model the problem with two philosophers and two forks only. In this case, the `fork1_free` and `fork2_free` are shared by the philosophers north and south, sitting at the North and South sides of the table respectively. This is modelled by the following Petri net.




Capture this Petri net within Workcraft as follows:

1. Build a Petri net model for a single dining philosopher.
2. Select the places and transitions representing the philosopher (exclude the fork1\_free and fork2\_free places) and combine them using the paging tool . The created *page* is our philosopher model.
3. Make use of copy-paste features to replicate the philosopher model. Name one of the pages south and the other north.
4. Rotate the north page using the rotation tool  or . Position the north and south pages opposite each other.
5. Share fork1\_free and fork2\_free places between the south and north philosophers by connection their take\_\*\_fork and put\_\*\_fork transitions accordingly.

Note the difference between groups and pages.

- Groups are just unnamed decorations for several nodes, while pages are named nodes containing other nodes. The nodes in different groups still should have unique names.
- Pages provide namespaces for their included nodes. Therefore nodes in different pages can have the same local name.

## Simulation

Enter the simulation tool by pressing the  button and play with the dining philosophers model. Initially both

forks are on the table and places `fork1_free` and `fork2_free` are marked. Each philosopher can take any of the forks - the `take_left_fork` and `take_right_fork` transitions are enabled for both north and south philosophers.

If you allow the same philosopher to take both forks (e.g., by clicking transitions `south/take_left_fork` and `south/take_right_fork`), then he can eat, while the other philosopher thinks. When he is finished with the meal he enters the thinking state (click transition `south/start_thinking`) and returns the forks (click transitions `south/put_left_fork` and `south/put_right_fork`). This trace of transitions brings us to the initial state where the philosophers compete for the forks again.

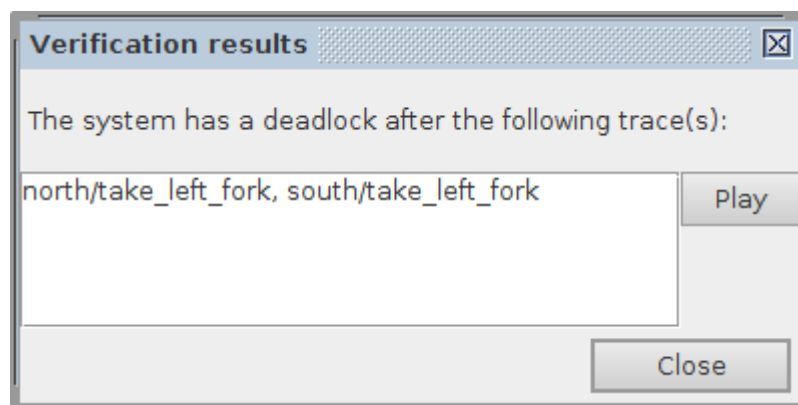
Another scenario is possible when each philosopher takes only one of the forks (e.g. click transitions `south/take_left_fork` and `north/take_left_fork`). This leads to a deadlock as the philosophers wait for each other to return the fork, but none of them can do it until they eat.

Simulate the above distinctive scenarios for dining philosophers. Explore the navigation buttons in the *Control panel* and go back and forth through the trace of events in your simulation. Try random simulation at different animation speeds.

## Verification

Various properties of a Petri net model can be verified using *Punf* and *MPSat* as the backend tools (see overview of [third-party tools](#) for more details). The tool interfaces available for Petri net verification are collected under *Verification* menu.

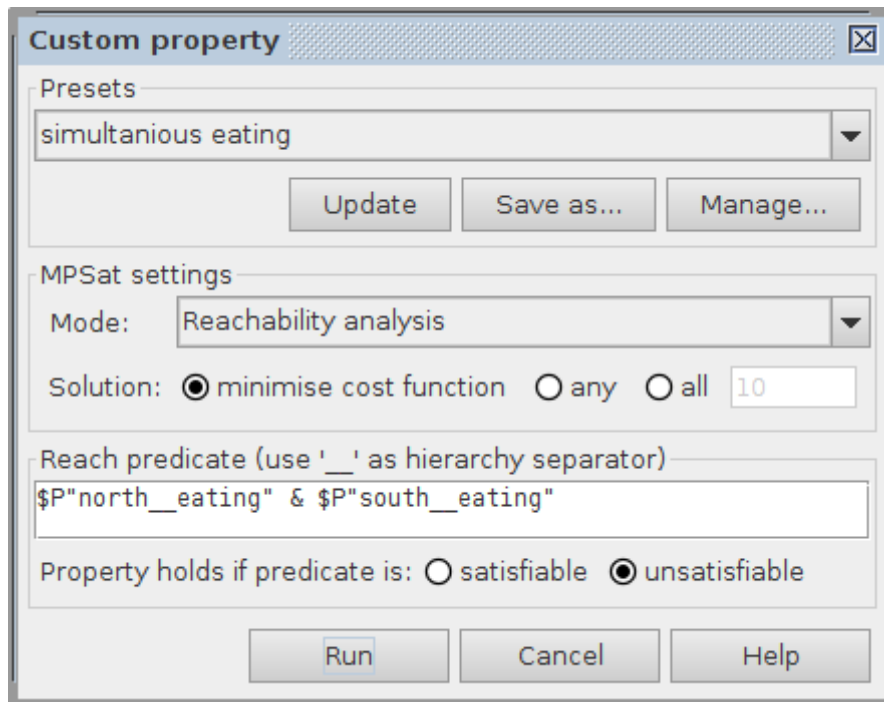
The easiest and most obvious property to check is that the model does not have a deadlock – choose the *Verification*→*Deadlock [MPSat]* menu item for this. As our dining philosophers model is not deadlock-free, this should result in the following error message with a trace leading to the deadlock (you can analyse the trace in the simulator by clicking the *Play* button).



Verification of custom properties is based on reachability analysis on the Petri net. For this a custom property should be expressed in [Reach language](#). For example, you can try and verify that both philosophers cannot be eating at the same time:

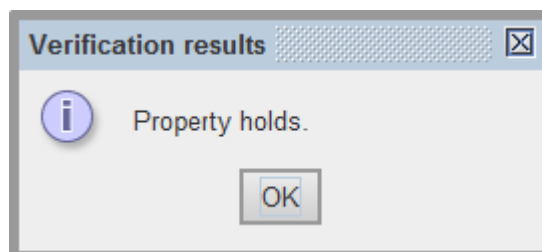
1. Open *Custom property* window via *Verification*→*Custom property [MPSat]*... menu.
2. In *MPSat settings* section set the *Mode* into *Reachability analysis* and the *Solution* into *minimise cost function*.
3. Enter the following Reach expression as the property predicate: `$P"north__eating" & $P"south__eating"`. Note that `__` (two underscores) is used as a hierarchy separator in Reach expression.
4. Select *unsatisfiable* to denote that the property holds if predicate is unsatisfiable.
5. Save this property as a preset for future use (*Save as...* button), e.g. under the name *simultaneous eating*.

The result should be similar to the following screenshot.



Let us explain how to interpret the Reach predicate. Consider an expression  $\$P$ "place\_name". It instructs MPSat to check if a marking is reachable where the place place\_name has a token. In basic cases one can apply logical operations and quantifiers (e.g.  $\&$ ,  $|$ , forall, existis, etc.) to a marking which violates the property. For expressing more sophisticated properties refer to [Reach language documentation](#).

In our case, the expression  $\$P$ "north\_\_eating" &  $\$P$ "south\_\_eating" tells MPSat to check if a state is reachable where both places north/eating and south/eating are marked with tokens, which corresponds to both philosophers eating at the same time. If such a marking is reachable, then our property is violated and a trace of Petri net transitions leading to this state is reported. If this Petri net marking is not reachable, then the property holds, as shown in the following screenshot.



Note that the Reach expression is usually formulated as the negation of the property one wants to check. If there is no reachable state satisfying this negation then the property itself is satisfied by all the reachable states.

Complete the following tasks:

1. Verify the model of dining philosophers for deadlocks. Simulate the trace leading to the deadlock.
2. Modify the model in such a way that the deadlock is prevented.
3. Verify that none of the philosophers can be eating with one of the forks still on the table.