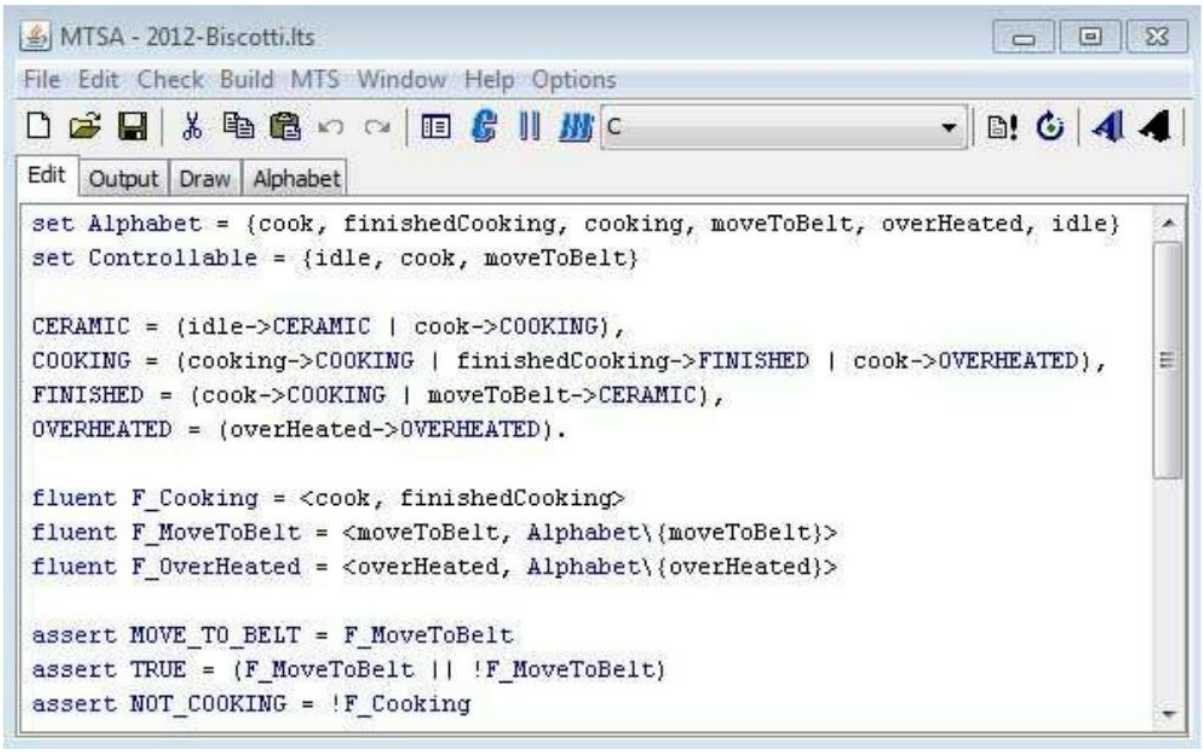
MTSA: User guide

1. Synthesising a controller

For synthesising a controller we need to model the system environment assumptions and the system goals.

1.1. Modelling the Environment

In MTSA environment models are described with an extension of the FSP (Finite State Process) The extension of FSP provided by the MTSA tool includes traditional operators for describing behaviour models, such as action prefix (->), choice (|), sequential composition (;), and parallel composition (||).
Our tool integrates functionality to construct, analyse and elaborate LTS models and provides a graphical environment aimed to facilitate these tasks. In our tool, environment models are described in FSP. In the picture below, we show a snapshot of the FSP editor provided by our tool.

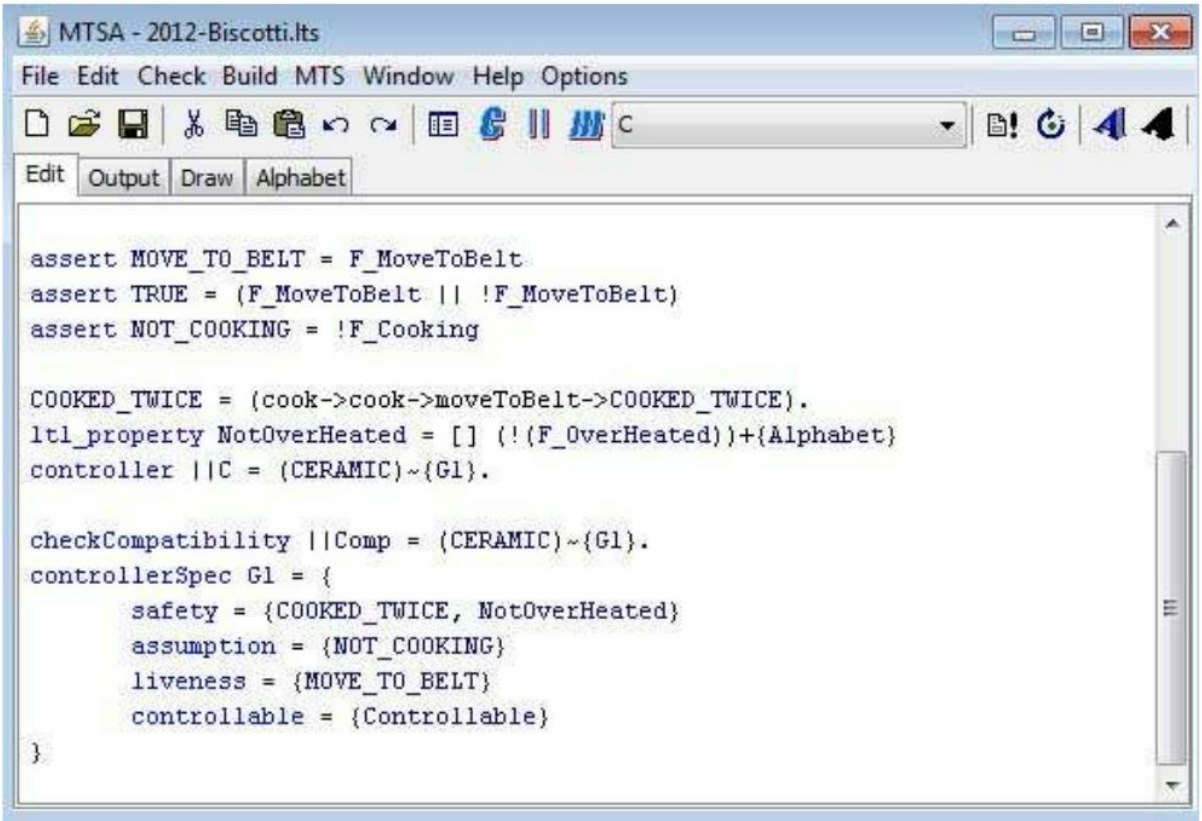


The code shown corresponds to a ceramic cooking example presented. It defines a process called CERAMIC that allows for either staying idle (looping to itself), or starting to cook, leading to the auxiliary process COOKING . Similarly, COOKING may loop to itself while the oven is cooking, or it may lead to the auxiliary process OVERHEATED if there is a second attempt to cook while the oven is still cooking. Additionally, if the oven signals that it has finished cooking the model transitions to FINISHED. Finally, FINISHED transitions either to

COOKING by starting to cook, or it resets to CERAMIC by moving the cooked ceramic to the conveyor belt.

1.2. Modelling Controller Goals

We have added a set of keywords to FSP to support our solution for dynamic update controllers. In the picture below we show the FSP code for synthesising a controller for the ceramic cooking example.



The controller operator returns (if exists) a controller that satisfies a specification for a given environment model. For example, we show the FSP code modelling the controller goals for the ceramic cooking example where controller is applied to the environment model CERAMIC and the goal G1 .

Controller goals are defined within the `controllerSpec` operator. The `safety` keyword allows for defining safety requirements. There are two ways of defining such restrictions. First, by providing an LTS model representing expected behaviour (e.g. COOKED_TWICE). Second, using the `ltl_property` operator which given a safety formula φ builds an LTS model E in which any trace violating φ leads to the error state in E (e.g. TOOL_ORDER).

The liveness part of the controller goal is defined using the assumption and liveness operators that specify the environment assumptions and controller liveness goals respectively. Both are defined with FLTL assertions represented with the keyword assertion. In the example, NOT_COOKING it is defined as the only environment assumption and, MOVE_TO_BELT as the only controller liveness expectation. Thus, the synthesised

controller will guarantee that if infinitely many times the oven is not cooking, objects are placed in the belt infinitely often.

1.3. Running the synthesis

Once the environment and the goal is modeled we have to specify the synthesis control problem. For this purpose, we need to write the following statement.

```
controller ||C = CERAMIC~{G1}.
```

where C is the name that the resultant controller will have after synthesising.
For running the synthesis, just look for C in the dropdown menu on top of the MTSA window.
If we want to get the parallel composition between C and E (C||E), we will write the following:

||Parallel = (C || CERAMIC).

2. Synthesising a Dynamic Update Problem

2.1 Modelling a Dynamic Update Problem

In the paper presented, we defined the inputs for a Dynamic Controller Update Synthesis problem. The MTSA tool allows the specification for solving this problem with the following statement:

```
UpdatingController UpdCont = {
   oldController = E||C : Process,
   oldEnvironment = E : FSP,
   hatEnvironment = Ê : FSP, // Most often E
   newEnvironment = E' : FSP,
   oldGoal = G : controllerSpec
   newGoal = G' : controllerSpec
   transition = T : ltl_property
   nonblocking,
   updateFluents = Q : Set
}
```

The elements in red determines each input represented in the paper presented. For instance, in `oldController` statement we should write the name of a process that is the parallel composition of the old environment and controller.
The elements in blue shows the kind of object that the input should be. For example, the input for the `oldGoal` must be a `controllerSpec`.

Then, to run the updating controllers solution we have to select the `UpdCont` target in the dropdown menu in MTSA.

2.2. Output for the Dynamic Update Control Problem

The output for any update problem is a big LTKS, thus, is probable that the draw tab does not show anything. Fortunately, we can animate the output by writing the following statement.

```
||UPDATE_CONTROLLER = UpdCont.
```

And running the parallel composition for `UPDATE_CONTROLLER`. (search this name in the dropdown menu after parsing the file).
For animating the controller just press the A button in top of the MTSA window. The pop up menu show all the events that can be executed at any state.
Also, we can check any defined `ltl_property` to the output by using the menu on top of the MTSA window.

```
Check >> LTL property >> "property name"
```