Evaluation Study

Tom Mens, Alexandre Decan

1 Preliminary questions

Please answer the following questions **before** starting the experiment!

1.1 Personal background

- Gender: ☐ Male ☐ Female ☐ Other:
- Age:
- Obtained degrees (diplomas) of higher education:
  - –
  - –
  - –

1.2 Programming experience

- How many years of programming experience do you have?
  ☐ None ☐ < 1 ☐ $1 - 2$ ☐ $3 - 4$ ☐ 5 or more
- How many years of experience do you have with Python programming?
  ☐ None ☐ < 1 ☐ $1 - 2$ ☐ $3 - 4$ ☐ 5 or more
- How would you assess your proficiency with Python programming?
  ☐ Inexistent ☐ Poor ☐ Acceptable ☐ Good ☐ Very Good
- How familiar are you with the following software development notions?

	I don't know what this is	I know it but never used it	I use it occasionally	I use it regularly
Behaviour-Driven Development (BDD)				
The Gherkin language				
Test-Driven and Test-First Development				
Unit testing				
Design by Contract				

1.3 Modeling experience

- In which year did you attend an UML modelling course? (either as part of your studies, or in another context)

- How would you assess your proficiency with **UML modelling**?
  ☐ Inexistent ☐ Poor ☐ Acceptable ☐ Good ☐ Very Good

- How familiar are you with the following software modelling notions? Add a × in the corresponding column in each line of the table.

	I don't know what this is	I know it but never used it	I use it occasionally	I use it regularly
Use case modelling				
Class diagrams				
Statecharts				
Component diagrams				

- How familiar are you with the following statechart concepts? Add a × in the corresponding column in each line of the table.

	I don't know what this is	I know it but never used it	I use it occasionally	I use it regularly
States				
Transitions				
Events				
Guards (conditions)				
State entry and exit actions				
Composite states				
Orthogonal regions of a composite state				

2 Preliminaries

Installing Python

For this experiment you need to have a working Python 3.4+ environment. While most Linux distributions come with both Python 2 and Python 3, it is possible that the default interpreter (i.e., the one that is executed when you type python in your console) is Python 2. You can check this by executing `python -V`. If Python 2 is the default interpreter, you should check if Python 3 is also available on your system, e.g., by using `python3 -V`. If Python 3 is available but not your default interpreter, you will need to replace python by python3 each time you need to execute Python. If Python 3 is not yet installed on your machine, you should follow the instructions from the official Python website to install it on your system:
`https://wiki.python.org/moin/BeginnersGuide/Download`

Installing Sismic

You also need a working copy of sismic, a research prototype tool that provides support for interpreting and testing statecharts in Python. Sismic can easily be installed using pip , the default package manager of Python (or pip3 if Python 3 is not your default interpreter for Python), for example: `pip3 install sismic --user`
The extra `--user` parameter is mandatory if you are not an administrator of your system, or if you don't want Sismic to be installed system-wide.

To check if Sismic is properly installed, execute the following command in your shell:
`python -c "import sismic; print(sismic.__version__)"`
This should display "*0.22.11*" in the terminal.

After the experiment, you will be able to easily uninstall Sismic from your system by simply executing `pip uninstall sismic`.

3 Experiment

The goal of the experiment is to adapt and evolve a statechart-based implementation of a microwave oven simulation to a more advanced version. The specification, functionalities and design of a basic microwave oven are provided in the following sections, together with specific tasks to get acquainted with the specification and design of the simulator.

Important note to participants: For most of the tasks in this experiment you are required to produce or modify some files that will be provided to you at the beginning of the task. **At the end of each task, store the files that you have produced in a separate folder, and label this folder with the task number** E.g., folder `4` would contain the files that you have produced as the result of task `4`. If you need to continue working on the same file in a later task, you need to copy this file first in a separate folder!

Important note! The tasks should be carried out ONE AT A TIME. You are **NOT** supposed to read the complete document at the start of the experiment, as reading later tasks may influence the results of earlier tasks!

Task 1: UI of a basic microwave oven simulator

Figure 1 shows a screenshot of the user interface (UI) of a basic microwave oven that is controlled by a statechart. The UI is implemented in a Python file microwave.py that can be executed using `python microwave.py microwave.yaml` where microwave.yaml is the path to the statechart specification file. (Remember that if Python 3 is not the default interpreter installed on your system, then you need to replace `python` by `python3` in every command.)

The UI contains three input buttons timer +, timer - and timer 0 to increment, decrement or reset to zero the cooking time, a well as a button start to start heating. The sensors of the microwave are also represented by buttons in the oven simulator: pushing tick advances the time with one second; while open door and close door allow to open and close the oven door, respectively. For convenience, the tick button is automatically pressed each second unless you uncheck the Auto-tick checkbox. On the right, the UI also exhibits the current state of the **Door**, **Heating** and **Beeper** components.

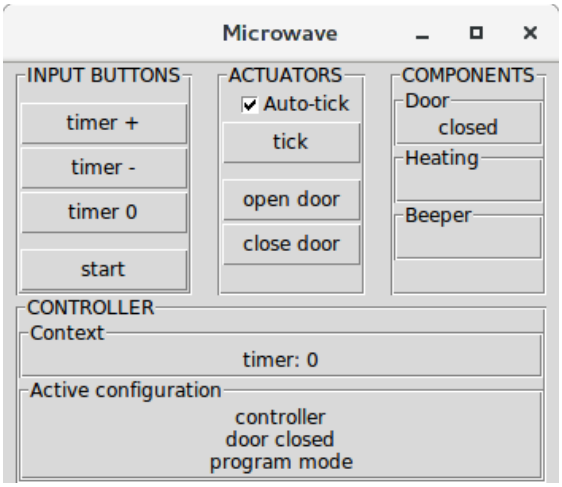

Figure 1: User interface of a simple microwave oven simulator

Task: Run this UI to get acquainted with the functioning of the microwave oven simulator. Indicate how many minutes you spent on this task.

Responses for task 1:

- Time (in minutes) spent on this task: _____
- How helpful was the provided UI for helping you to understand the functional behaviour of the oven simulator?
  ☐ Not at all ☐ Slightly ☐ Moderately ☐ Very ☐ Extremely
- Did you observe any unexpected functional behaviour of the oven simulator while using the UI? Yes: ☐ No: ☐
  If Yes, please detail:

  _____

  _____

  _____

  _____

  _____

- **(Optional)** Provide any other comments or feedback you would like to share with us w.r.t. this task.

  _____

  _____

  _____

  _____

  _____

Task 2: Reading BDD scenarios of the oven's functional behaviour

Behaviour-Driven Development (BDD) is an extension of Test-Driven Development (TDD). It allows non-programmers, such as requirement engineers, to use their native language in combination with the language of domain-driven design to write executable functional test cases. This allows them to specify how the software should behave, without needing to focus on the technical details. Developers can easily convert these functional test cases into executable code using a language-specific framework.

Requirement engineers can use the human-readable Gherkin language to describe functional software features by means of representative scenarios of expected outcomes. Such features serve as automated tests that can be executed to check the expected outcome. As an example, the functional behaviour of the basic microwave oven is expressed using BDD feature specifications in Figure 2 and Figure 3.

The documentation for the Gherkin language can be found at `https://github.com/cucumber/cucumber/wiki/Gherkin`.

Task 2: Read the scenarios of Figures 2 and 3 attentively. Indicate how many minutes you spent on this task.

Responses for task 2:

- Time (in minutes) spent on this task: _____
- How understandable are the provided feature specifications and scenarios according to you?
  ☐ Not at all ☐ Slightly ☐ Moderately ☐ Very ☐ Extremely
- Are you confident that the microwave oven simulator satisfies the provided scenarios?
  ☐ Not at all ☐ Slightly ☐ Moderately ☐ Very ☐ Extremely
- Do you feel that some important scenarios are still missing? Which ones?

  _____

  _____

  _____

  _____

```
Feature: Basic microwave

  Background: Place an item into oven
    Given I open the door
    and I place an item in the oven
    and I close the door

  Scenario: Timer value is kept if door is opened then closed
    Given I press increase timer button 5 times
    and I open the door
    and I close the door
    and I press start button
    when 5 seconds elapsed
    then magnetron stops heating

  Scenario: Timer value is kept if door is opened then closed while cooking
    Given I press increase timer button 5 times
    and I press start button
    and 2 seconds elapsed
    when I open the door
    and I close the door
    and I press start button
    and 3 seconds elapsed
    then magnetron stops heating

  Scenario: Timer must be set to start cooking
    When I press start button
    then magnetron does not start heating

  Scenario: Timer must be greater than 0 to start cooking
    Given I press increase timer button
    and I press decrease timer button
    when I press start button
    then magnetron does not start heating

  Scenario Outline: Cooking time must correspond to timer
    Given I press increase timer button <time> times
    and I press start button
    when <seconds> seconds elapsed
    then magnetron stops heating

    Examples:
      | time | seconds |
      | 1    | 1       |
      | 2    | 2       |
      | 10   | 10      |
      | 50   | 50      |
```

Figure 2: Content of microwave.feature, BDD feature specification with executable scenarios of basic microwave oven. (Part 1)

```
49    Scenario: Cooking time can be reset
50      Given I press increase timer button 5 times
51      and I press reset timer button
52      and I press increase timer button 2 times
53      and I press start button
54      when 2 seconds elapsed
55      then magnetron stops heating
56
57    Scenario: Cooking time can be reset while cooking
58      Given I press increase timer button 5 times
59      and I press start button
60      and 2 seconds elapsed
61      when I press reset timer button
62      then magnetron stops heating
63
64    Scenario Outline: Timer can be increased while cooking
65      Given I press increase timer button <initial> times
66      and I press start button
67      and <elapsed> seconds elapsed
68      and I press increase timer button <additional> times
69      when <remaining> seconds elapsed
70      then magnetron stops heating
71
72      Examples:
73        | initial | elapsed | additional | remaining |
74        | 10      | 6       | 1          | 5         |
75        | 10      | 0       | 5          | 15        |
76        | 10      | 6       | 5          | 9         |
77
78    Scenario Outline: Timer can be decreased while cooking
79      Given I press increase timer button <initial> times
80      and I press start button
81      and <elapsed> seconds elapsed
82      and I press decrease timer button <substracted> times
83      when <remaining> seconds elapsed
84      then magnetron stops heating
85
86      Examples:
87        | initial | elapsed | substracted | remaining |
88        | 10      | 6       | 1           | 3         |
89        | 10      | 0       | 6           | 4         |
90        | 10      | 6       | 3           | 1         |
91        | 10      | 6       | 4           | 0         |
```

Figure 3: Content of microwave.feature, BDD feature specification with executable scenarios of basic microwave oven. (Part 2)

Task 3: Executing BDD scenarios of the oven's functional behaviour

Sismic provides an easy way to automatically execute the scenarios, by using the `sismic-behave` command. If, for any reason, the `sismic-behave` command-line utility is not available on your system, you can replace it by `python -m sismic.testing.behave` The command requires some additional parameters: a positional `path` argument that corresponds to the path of the statechart ; a `--steps` argument that corresponds to the Python file that implements the steps used in a .feature file (must be set to `steps.py` for this experiment) ; a `--features` argument that corresponds to a list of space-separated paths, each of them being a .feature file.[1]

For instance, the following command should execute the aforementioned microwave oven scenarios:

```
sismic-behave microwave.yaml --steps steps.py --features microwave.feature
```

Task 3: Execute the scenarios of Figures 2 and 3 to convince yourself that the implementation complies to the intended functionality. Indicate how many minutes you spent on this task.

[1] There also exists an optional `--debug-on-error` argument that drops a pdb (or ipdb if available) debugger in case of error. This should be used only if you are acquainted with both Python and its debugger.

Responses for task 3:

- Time (in minutes) spent on this task: _____
- Were you able to execute the scenarios successfully? Were you satisfied by the output returned by the scenario runner?

_____

_____

_____

_____

- How did your confidence in the functional behaviour of the microwave oven simulator change after the automatic execution of the scenarios?

  ☐ Much lower confidence ☐ Lower ☐ About the same ☐ Higher ☐ Much higher

- Having used this tool for automatic execution of human-readable functional scenarios, what is your opinion of this way of specifying and verifying executable requirements?

_____

_____

_____

_____

- **(Optional)** Provide any other comments or feedback you would like to share with us w.r.t. this task.

_____

_____

_____

Task 4: Specification of the oven controller statechart

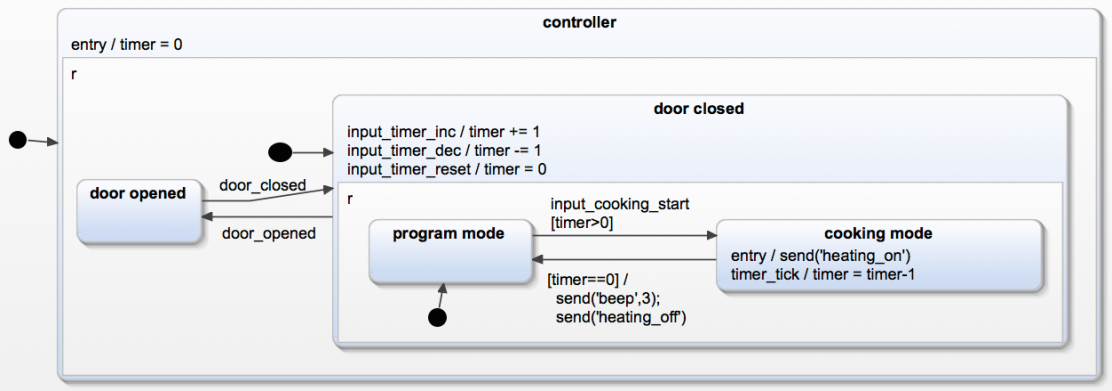

Figure 4: Statechart of a simple microwave oven controller

The UI of Figure 1 interacts with the statechart visualised in Figure 4. The UI maps buttons pressed in the UI to events received by the statechart according to the mapping of Table 1.

Table 1: Mapping of UI buttons to incoming statechart events

UI button	Event name
timer +	**input_timer_inc**
timer -	**input_timer_dec**
timer 0	**input_timer_reset**
start	**input_cooking_start**
tick	**timer_tick**
open door	**door_opened**
close door	**door_closed**

Conversely, events raised by the statechart will control some of the oven's components. This basic microwave has a **Heating** component, representing the actual magnetron that can receive **heating_on** and **heating_off** events to start or stop emitting microwaves. It also has a **Beeper** component that can receive **beep** events (parametrised by an integer value) to emit a sound.

The complete statechart of the microwave is provided in Figure 5, written in a human-friendly textual YAML notation. The statechart notation is described in the Sismic

documentation:
`http://sismic.readthedocs.io/en/master/format.html`.

```
statechart:
  name: Microwave controller
  description: |
    in:
     input_timer_inc
     input_timer_dec
     input_timer_reset
     input_cooking_start
     timer_tick
     door_opened
     door_closed

    out:
     heating_on
     heating_off
     beep(number:int)
  preamble: |
      timer = 0
  root state:
    name: controller
    initial: door closed
    states:
      - name: door closed
        transitions:
          - event: door_opened
            target: door opened
          - event: input_timer_inc
            action: timer += 1
          - event: input_timer_dec
            action: timer -=1
          - event: input_timer_reset
            action: timer = 0
        initial: program mode
        states:
          - name: program mode
            transitions:
              - event: input_cooking_start
                guard: timer > 0
                target: cooking mode
          - name: cooking mode
            on entry: |
              send('heating_on')
            transitions:
              - guard: timer == 0
                target: program mode
                action: |
                  send('heating_off')
                  send('beep', number=3)
              - event: timer_tick
                action: timer -= 1
      - name: door opened
        transitions:
          - event: door_closed
            target: door closed
```

Figure 5: microwave.yaml file containing the oven controller statechart.

Task: Inspect this YAML file to get acquainted with how statecharts are textually represented. Indicate how many minutes you spent on this task.

Responses for task 4:

- Time (in minutes) spent on this task: _____
- How readable is the visual representation of the statechart in Figure 4 according to you?
  ☐ Not at all ☐ Slightly ☐ Moderately ☐ Very ☐ Extremely
- How readable is the textual YAML specification of the statechart in Figure 5 according to you?
  ☐ Not at all ☐ Slightly ☐ Moderately ☐ Very ☐ Extremely
- Do you think it would be easy to define and modify statecharts using the visual representation only?
  ☐ Very easy ☐ Easy ☐ Difficult ☐ Very difficult
- Do you think it would be easy to define and modify statecharts using the textual YAML specification only?
  ☐ Very easy ☐ Easy ☐ Difficult ☐ Very difficult
- **(Optional)** Provide any other comments or feedback you would like to share with us w.r.t. this task.

  _____

  _____

  _____

  _____

  _____

Task 5: Specifying contracts in a statechart

The oven controller statechart presented visually in Figure 4 and textually in Figure 5 does not impose any lower or upper bound on the value of the timer variable. Ideally, the timer value should range between zero seconds and one hour. In any case, the timer should never get a negative value. One way to impose such constraints is through the use of contracts.

Sismic allows contracts to be defined in a statechart, both on transitions and on states, as preconditions, postconditions and invariants. Contracts must be directly specified using the YAML notation, as explained in the following documentation: `http://sismic.readthedocs.io/en/master/contract.html`.[2]

By default, the GUI provided in microwave.py does not check whether contracts are satisfied or not during the execution. Contracts checking can be enabled by appending `--contracts` on the command-line: `python microwave.py microwave.yaml --contracts`

For convenience, the UI displays the name and the value of each internal variable that is used by the statechart (e.g., the `timer` variable), and also displays the current active configuration, i.e., the set of currently active states. During the execution of the statechart, the resulting steps (events that are consumed, events that are sent, states that are entered and states that are exited) are all displayed in the terminal that launched the UI.

[2]The subsection about sequential conditions can reasonably be ignored for this experiment.

Task:

(a) Add contracts (invariants) to the YAML specification of the statechart to assert that the timer value should never be negative and never exceed one hour (3600 seconds).

(b) Use the GUI of Figure 1 to raise violations of these contracts.

(c) Modify the YAML specification of the statechart so that the contracts can never be violated.

Responses for task 5:

- Provide the version of your modified statechart YAML file after step (c).
- Time (in minutes) spent on this task: _____
- How useful is the notion of contracts for statecharts according to you?
  ☐ Not at all ☐ Slightly ☐ Moderately ☐ Very ☐ Extremely
- How convenient is the implementation of contracts provided by Sismic?
  ☐ Very inconvenient ☐ Inconvenient ☐ Convenient ☐ Very convenient
- **(Optional)** Provide any other comments or feedback you would like to share with us w.r.t. this task.

_____

_____

_____

_____

_____

Task 6: Adding Lamp behaviour to the microwave oven

Modify your statechart specification so that it controls an additional **Lamp** component that can either be turned on or off by sending events **lamp_switch_on** and **lamp_switch_off**, respectively. The lamp should be on while the oven door is open. While the oven door is closed, the lamp should only remain on during heating (i.e., while the microwaves are being emitted).

For convenience, to carry out the following taks we provide you with a new GUI that integrates the additional **Lamp** component.

Task: Follow a test-driven process to add lamp functionality to your statechart:

(a) Create a .feature file to specify the executable scenarios corresponding to the intended lamp functionality. Table 2 lists all predefined steps that you can use when specifying your new executable scenarios.

Table 2: List of predefined BDD steps in steps.py

Buttons and actuators	**Assertions**
I open the door	magnetron starts heating
I close the door	magnetron stops heating
I place an item in the oven	magnetron does not start heating
I remove the item from the oven	magnetron does not stop heating
I press increase timer button	oven beeps once
I press increase timer button {time} times	oven beeps {time} times
I press decrease timer button	lamp turns on
I press decrease timer button {time} times	lamp turns off
I press reset timer button	lamp does not turn off
I press start button	lamp does not turn on
I press stop button	
{second} second elapsed	
{second} seconds elapsed	
I do "{scenario}"	
scenario "{scenario}"	

(b) Execute the scenarios of step (a) on your statechart, and verify that that these scenarios fail (because the lamp functionality is not implemented yet).

(c) Modify the statechart to accommodate the intended lamp functionality.

(d) Rerun the scenarios that you specified in step (b) on the statechart of step (c). If one of the scenarios failed, repeat the process until all scenarios succeed.

Responses for task 6:

- Provide the .feature file containing your own scenarios.
- Provide the version of your modified statechart YAML file.
- List all scenario failures that you encountered throughout the process, and explain why they occurred.
  - –
  - –
  - –
  - –
- Time (in minutes) spent on this task (including the time needed to respond to the previous question): _____
- How easy was it for you to specify your scenarios?
  ☐ Very easy ☐ Easy ☐ Difficult ☐ Very difficult
- How easy was it for you to add the requested functionality to the statechart?
  ☐ Very easy ☐ Easy ☐ Difficult ☐ Very difficult
- How helpful were the scenarios for implementing the requested statechart functionality correctly?
  ☐ Not at all ☐ Slightly ☐ Moderately ☐ Very ☐ Extremely
- **(Optional)** Provide any other comments or feedback you would like to share with us w.r.t. this task.

  _____

  _____

  _____

  _____

  _____

Task 7: Verifying Lamp behaviour based on external scenarios

Consider the BDD scenarios of Figure 6 for a microwave statechart containing the lamp functionality.

```
Feature: Lamp

  Scenario: Lamp turns on when door is opened
    When I open the door
    then lamp turns on

  Scenario: Lamp turns off when door is closed
    Given I open the door
    when I close the door
    then lamp turns off

  Scenario: Lamp turns on when cooking starts
    Given I open the door
    and I place an item in the oven
    and I close the door
    and I press increase timer button 5 times
    when I press start button
    then lamp turns on

  Scenario: Lamp stays on when door is opened while cooking
    Given scenario "Lamp turns on when cooking starts"
    and 2 seconds elapsed
    when I open the door
    then lamp does not turn off

  Scenario: Lamp turns off when cooking time elapsed
    Given scenario "Lamp turns on when cooking starts"
    when 5 seconds elapsed
    then lamp turns off
```

Figure 6: File lamp.feature

Task: Apply the BDD scenarios of Figure 6 (provided in file lamp.feature to your microwave statechart containing the lamp functionality. If one of the scenarios fails, continue to modify your statechart until all scenarios succeed. (Keep track of the failed scenarios in your responses to this task.)

Responses for task 7:

- Provide the version of your modified statechart YAML file (if any modifications were needed to make the scenarios succeed).
- List all scenario failures that you encountered throughout the process, and explain why they occurred.
  - _____
  - _____
  - _____
  - _____
  - _____
  - _____
- Time (in minutes) spent on this task (including the time needed to answer the previous questions): _____
- **(Optional)** Provide any other comments or feedback you would like to share with us w.r.t. this task.

_____

_____

_____

_____

_____

Task 8: Verifying safety properties over a running microwave

Any microwave oven is supposed to satisfy an important safety requirement: no microwaves should be emitted while the oven door is open. The **Heating** component (a.k.a. the magnetron) is in charge of emitting microwaves. The microwave controller statechart implicitly assumes that, whenever a **heating_on** event is sent, the microwaves start emitting, and whenever a **heating_off** event is sent, the microwaves stop emitting. Under these assumptions, we can express and verify two undesirable properties on the controller statechart. These properties will be expressed through the use of so-called *property statecharts*.

A property statechart defines a property that should (or should not) be satisfied by other statecharts. A property statechart is like any other statechart, in the sense that neither their syntax nor their semantics differs from any other statechart. The difference comes from the events it receives and the role it plays. For this experiment, we consider that if the run of the property statechart ends in a final state, then the property is considered as unsatisfied. The first property statechart is shown visually in Figure 7 and textually in the YAML file of Figure 8. It verifies that heating does not happen while the door is still open. If a **heating_on** event is sent before the **door_closed** event is received (after having received a **door_opened** event first), the property statechart will go to its final state, indicating a failure of the property and hence a breach of the safety requirement.

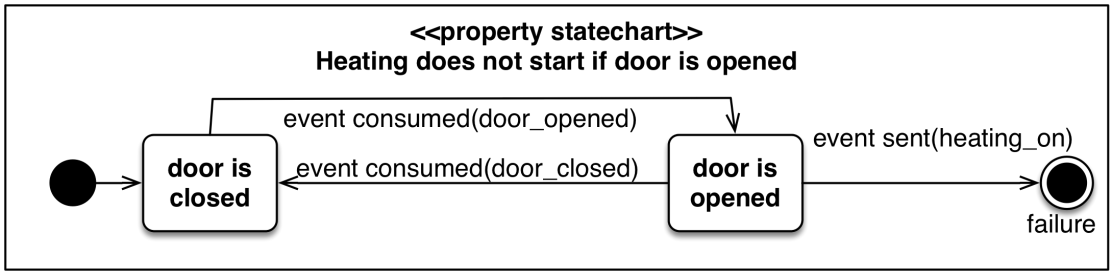

Figure 7: Property statecharts expressing the *undesirable* property that no heating should occur while the door is still open.

A property statechart expects specific events to be sent to the statechart. For instance, an **event consumed** event is sent to the property statechart each time an event **x** is consumed by the statechart under test. The name **x** of the event can be accessed through `event.event.name` (the first `event` refers to the **event consumed** event, the second `event` refers to the **x** event). Similarly, an **event sent** event is sent to the property statechart each time an event is sent by the statechart under test. Other events that are automatically sent to the property statechart are **state entered**, **state exited**, **transition processed**, ... The complete list of supported events, and their parameters, is available at `http://sismic.readthedocs.io/en/master/testing.html`.

For convenience, the provided microwave.py script can be configured to execute and check property statecharts during the execution of the GUI. This can be done by pro-

```
statechart:
  name: Cooking does not start if door is opened
  root state:
    name: root
    initial: door is closed
    states:
      - name: door is closed
        transitions:
          - target: door is opened
            event: event consumed
            guard: event.event.name == 'door_opened'
      - name: door is opened
        transitions:
          - target: door is closed
            event: event consumed
            guard: event.event.name == 'door_closed'
          - target: failure
            event: event sent
            guard: event.event.name == 'heating_on'
      - name: failure
        type: final
```

Figure 8: YAML file of the property statechart

viding an additional parameter `--properties` followed by a list of paths. For example, the first property statechart can be automatically checked by running:

```
python microwave.py microwave.yaml --properties property1.yaml
```

Task:

(a) Use the microwave GUI, while running the first property statechart property1.yaml in the background, and verify if the safety property is not violated.

(b) Write a YAML file property2.yaml corresponding to the following property: "the oven should stop emitting microwaves while the door is opened". This property can be verified by checking that the **heating_off** event is sent by the statechart sufficiently rapidly (i.e., before the next **timer_tick** event is consumed). If a **timer_tick** has been consumed before **heating_off** is sent, the property statechart will go to its final failure state.

(c) If the property can be violated, correct the statechart until the property can no longer be violated.

Responses for task 8:

- List all property violations that you encountered throughout the process, and explain why they occurred.
  - _____
  - _____
  - _____
  - _____
  - _____
  - _____

- Provide your property statechart YAML file, labelled property2.yaml.
- If you needed to make corrections to the microwave statechart for this task, provide its new YAML file.
- Time (in minutes) spent on this task (including the time needed to answer the previous questions): _____
- How useful is the notion of property statecharts according to you?
  ☐ Not at all ☐ Slightly ☐ Moderately ☐ Very ☐ Extremely
- How easy was it for you to *understand* the property statechart presented in Figures 7 and 8?
  ☐ Very easy ☐ Easy ☐ Difficult ☐ Very difficult
- How easy was it for you to *write* your own property statechart?
  ☐ Very easy ☐ Easy ☐ Difficult ☐ Very difficult
- Were both property statecharts (the one that was provided and the one that you wrote) helpful to ensure the the statechart's safety requirements?
  ☐ Not at all ☐ Slightly ☐ Moderately ☐ Very ☐ Extremely

- **(Optional)** Provide any other comments or feedback you would like to share with us w.r.t. this task.

Task 9: Adding WeightSensor behaviour to the microwave oven

Modify your statechart specification so that it controls an additional **WeightSensor** that will detect when an item is placed in the oven (event **item_placed**) or removed from the oven (event **item_removed**).

For convenience, we provide you a new GUI that already contains two buttons place item and remove item that simulate the interaction between an item and the microwave. Whenever an item is placed in the oven, the **WeightSensor** reacts and automatically sends event **item_placed** to the statechart and, whenever an item is removed from the oven, it reacts by automatically sending event **item_removed** to the statechart. The UI also exhibits the state of the **WeightSensor** component, indicating whether an item is currently detected or not (see Figure 9, right).

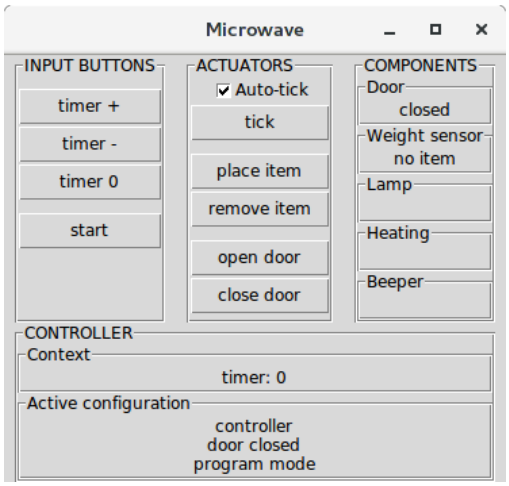

Figure 9: Modified user interface of a simple microwave oven simulator

Task: Follow a test-driven process to support weight sensor events in your statechart. In particular, it should only be possible to program the oven, and start heating, when an item is placed in the oven with closed door.
(a) Write BDD scenarios to specify the intended functional behaviour of a microwave oven with weight sensor.
(b) Modify the statechart to accommodate this new functionality.
(c) Check that the scenarios created in step (a) are satisfied.
(d) Check that all other existing scenarios (defined and used in previous tasks), remain satisfied after having added the weight sensor functionality in the statechart.
(e) Use the GUI to check that all existing contracts and property statecharts remain satisfied after having added the functionality in the statechart.

Responses for task 9:

- Provide the .feature file containing the scenarios created in step (a).
- Provide the version of the modified statechart YAML file in step (b).
- List all failures (of scenarios, contracts or property statecharts) that you encountered throughout the process, and explain why they occurred.

_____

_____

_____

_____

_____

- Time (in minutes) spent on this task: _____
- **(Optional)** Provide any other comments or feedback you would like to share with us w.r.t. this task.

_____

_____

_____

_____

_____

4 Post-Study Questions

Now that you have finished all tasks, please give your final appreciation of the three proposed mechanisms (BDD scenarios, statechart contracts, and property statecharts)?

- How likely are you to use the proposed mechanism for *creating a new statechart*?

	Not at all	Slightly	Moderately	Very	Extremely
BDD scenarios					
statechart contracts					
property statecharts					

- How likely are you to use the proposed mechanism for *verifying an existing statechart*?

	Not at all	Slightly	Moderately	Very	Extremely
BDD scenarios					
statechart contracts					
property statecharts					

- How likely are you to use the proposed mechanism for *modifying an existing statechart*?

	Not at all	Slightly	Moderately	Very	Extremely
BDD scenarios					
statechart contracts					
property statecharts					