University of St.Gallen

Institute of Computer Science

Web-based Autonomous Systems, FS2023
Danai Vachtsevanou, Andrei Ciortea
{firstname}.{lastname}@unisg.ch

Exercise 5: BDI Agents

Deadline: **March 28, 2023; 23:59 CET**

In this exercise, you will gain hands-on experience in programming BDI agents using Jason and AgentSpeak. You will:

1.) complete several warm-up programming exercises using the AgentSpeak language;

2.) complete a JaCaMo application by programming a BDI agent in AgentSpeak.

(1) Task 1 (3.5 points): Programming agents in AgentSpeak

Your first task is to complete the following programming exercises using the AgentSpeak language.

(2) Task 1.1 (1.6 points): An illuminance controller agent (`illuminance_controller.asl`[1]) has the design objective of maintaining the illuminance in a room at a given target level by controlling the blinds and the lights. Study and run the provided agent program to understand the agent's behavior (see the README here[2] for instructions on how to run the Gradle task `task1_1`).

Note that the agent is capable of performing multiple external actions exposed by artifacts in the room, namely: `turnOnLights`, `turnOffLights`, `raiseBlinds`, `lowerBlinds`. Additionally, the agent may hold beliefs about its environment, namely:

- the illuminance level in the room measured in luminous flux per area unit (lux); e.g., a current illuminance of 0 lux can be represented as: `current_illuminance(0)`;
- the state of the lights in the room, which can be turned on or off and represented as: `lights("on")`, `lights("off")`;
- the state of the blinds in the room, which can be raised or lowered and represented as: `blinds("raised")`, `blinds("lowered")`;
- the weather conditions, which can be sunny or cloudy and represented as: `weather("sunny")`, `weather("cloudy")`.

Study and run the provided agent program to understand the agent's behavior. Then, update the implementation as follows:

1.) Initially, the `current_illuminance` is 0 lux, and the `target_illuminance` is 400 lux. In order to increase the illuminance from 0 to 400, the agent turns on the lights. However, the program will then fail because the agent does not know how to handle the event `+!manage_illuminance` when the current illuminance is equal to the target illuminance. Write a plan that handles this case by printing a message that the design objective has been achieved.

[1] Illuminance controller agent: https://github.com/HSG-WAS-SS23/exercise-5/blob/main/src/agt/task1/illuminance_controller.asl

[2] Exercise README; Task 1.1: https://github.com/HSG-WAS-SS23/exercise-5#task-11

2.) Although the agent successfully maintains the illuminance at the target level, it does not achieve this using the most energy-efficient method, since, during sunny days, it can achieve the target level of 400 lux by raising the blinds instead of turning on the lights. Update the program so that the agent first tries to increase the illuminance by raising the blinds when the weather is sunny. You should now observe that initially the agent raises the blinds in sunny weather, but it also turns on the lights if the weather becomes cloudy.

3.) Now the agent successfully manages the illuminance in a more energy-efficient manner in both sunny and cloudy weather. However, if the agent increased the illuminance in the past by raising the blinds, and now the weather is cloudy, the blinds remain unnecessarily raised. Write a plan that enables the agent to react to the deletion of the belief `weather("sunny")` by lowering the blinds if the blinds are currently raised.

4.) Decrease the `target_illuminance` from 400 to 350 lux. Now the agent is stuck in a loop, turning on and off the lights because it cannot maintain the illuminance at the exact target level. Update the inference rules `requires_darkening` and `requires_brightening` so that the predicates are inferred as true only if the current illuminance differs by +-100 (respectively) from the target illuminance. The agent should now avoid the hysteresis phenomenon, considering that the illuminance is maintained close to the target level of 350 lux when either the blinds are raised or the lights are turned on.

(3) **Task 1.2 (1.9 points):** Complete the implementation of the agent program of a simple agent (`simple_agent.asl`[3], see the README here[4] for instructions on how to run the Gradle task `task1_2`):

1.) Write a plan for enabling the agent to achieve the goal of adding two numbers `X`, `Y`. For example, if `X` is 4 and `Y` is 2 the `Sum` should be 6. You can use the `+` operator.

2.) Write two plans for enabling the agent to achieve the goal of dividing two numbers. For example, if the `Dividend` is 4 and the `Divisor` is 2 the `Quotient` should be 2. The 1st plan should be applicable when the `Divisor` is equal to 0, and print that the division is not possible. Otherwise, the 2nd plan should be applicable, and it should perform the division. You can use the `/` operator.

3.) Write an inference rule for enabling the agent to infer that a number is even, and an inference rule to infer that a number is odd. For example, `even(4)` should be inferred as true, but `odd(4)` should be inferred as false. The opposite should be inferred for the number 5. You can use the `mod` operator.

4.) Write plans for enabling the agent to achieve the goal of adding integers within the range `[Start, End]` to a list. For example, for the range [0, 4], the `List` should contain the integers 0, 1, 2, 3, and 4. You are adviced to use recursion and the `|` list concatenation operator. You are provided with an example plan for printing lists with recursion and the list concatenation operator.

(4) **Task 2 (6.5 points) Programming a BDI agent**

Your second task is to complete a JaCaMo application by programming a BDI agent in AgentSpeak. The agent is a personal assistant (`personal_assistant.asl`[5]) whose design objective is to assist its user through their daily activities. Such is the case when the assistant believes that the

[3]Simple agent: https://github.com/HSG-WAS-SS23/exercise-5/blob/main/src/agt/task1/simple_agent.asl
[4]Exercise README; Task 1.2: https://github.com/HSG-WAS-SS23/exercise-5#task-12
[5]Personal assistant agent: https://github.com/HSG-WAS-SS23/exercise-5/blob/main/src/agt/task2/personal_assistant.asl

user is asleep, and that the user has an upcoming event to attend. Considering its options, the assistant strives to wake up the user, e.g. by turning on the lights, raising the blinds, or setting the mattress to a vibration mode.

The assistant can monitor the state of the user and its environment by observing multiple artifacts in the room; specifically, it can monitor:

- the upcoming events of the user through a calendar service; e.g., an upcoming event that is about to start can be represented as: `upcoming_event("now")`;
- the state of the user as monitored by the user's wristband; e.g., the state of the user being awake or asleep can be represented as: `owner_state("awake")`, `owner_state("asleep")`;
- the state of the lights in the room, which can be turned on or off and represented as: `lights("on")`, `lights("off")`;
- the state of the blinds in the room, which can be raised or lowered and represented as: `blinds("raised")`, `blinds("lowered")`;
- the state of the user's mattress which can be in an idle mode or a vibrations mode and represented as: `mattress("idle")`, `mattresss("vibrating")`.

Additionally, the assistant can perform multiple external actions exposed by the artifacts in the room; specifically it can control:

- the state of the lights in the room with the actions: `turnOnLights`, `turnOffLights`;
- the state of the blinds in the room with the actions: `raiseBlinds`, `lowerBlinds`;
- the state of the user's mattress with the actions: `setIdleMode`, `setVibrationsMode`.

Complete the implementation of the agent program `presonal_assistant.asl` to enable the agent to wake up the user in case there are any upcoming events (see the README here[6] for instructions on how to run the Gradle task `task2`):

1.) Write plans so that the assistant prints relevant messages when a belief about the states of devices (lights, blinds, mattress) or the owner (reported by the wristband) is added or deleted. When running the JaCaMo application, you can visit http://localhost:3272/ to inspect how these relevant beliefs look like, and use the available actions to update the device states.

2.) Write plans so that the assistant reacts to the addition of the belief `upcoming_event("now")`. If the user is awake, the assistant should print "Enjoy your event". If the user is asleep, the assistant should print "Starting wake-up routine".

3.) Update the agent program so that, when initialized, the assistant holds beliefs about how the user prefers to be woken up (the most preferred method is assigned the lowest rank):

- the user ranks waking up with artificial light with a 2.
- the user ranks waking up with natural light with a 1.
- the user ranks waking up with its mattress' vibration mode with a 0.

Additionally, write an inference rule that enables the assistant to infer what is the `best_option(Option)` for waking up the user based on the available rankings. E.g. based on the current rankings, only `best_option(vibrations)` should be inferred as true, while corresponding predicates for the other wake-up methods should be inferred as false.

4.) Write plans that enable the assistant to achieve the goal of waking up the user. The agent should wake up the user based on the current `best_option`. E.g. if `best_option(vibrations)`

[6]Exercise README; Task 2: https://github.com/HSG-WAS-SS23/exercise-5#task-2

is true, the agent should perform the action `setVibrationsMode`. The agent should continue striving for waking up the user for as long as the user is considered to be asleep, and print a message that the design objective have been achieved if the user is now considered to be awake. Update your implementation from step 2.) so that the agent creates the goal of waking up the user when there is an upcoming event and the user is asleep. You should observe that the agent attempts waking up the user by setting the mattress to the vibrations mode, but the user remains asleep.

5.) Update the agent program so that the agent takes note of any methods that it has already used in its attempt to wake up the user. Additionally, update the program so that the `best_option` is now selected only among the non-used methods. You should observe that the agent attempts to wake up the user first by setting the mattress to the vibrations mode, then, by raising the blinds, and finally, by turning on the lights. By employing all these methods, you should observe that the user is now awake.

(5) **Hand-in Instructions** By the deadline, hand in via Canvas a **zipfile** that contains:

1.) a PDF file that includes the link to the GitHub fork containing **code for Task 1 and Task 2**;

2.) any additional instructions for how to run your code (if non-obvious).