Lab 1: SLEEC DSL

This lab consists of three parts, each expected to take roughly 30 min.

Part 1: Modeling in SLEEC

Amie is a home assistant designed to assist you in your smart home, providing convenience and enhancing your living experience.

Below, we describe Amie system capabilities and its operating environment. We then provide a partial description of events and measures in SLEEC for Amie.

System capabilities:

- Collect User Consent:
  User access to the assistant is granted only after providing informed consent.
- Door Control:
  - Open and close doors.
  - Lock and unlock doors.
- Lighting Control:
    Turn lights on and off.
- Music Control:
  - Turn music on and off.
  - Switch songs.
  - Increase or decrease the volume.
- Curtain and Window Control:
  - Open and close curtains.
  - Open and close windows.

Environment
- Apartment Layout:
  - Kitchen
  - Bedroom
  - Living room
  - Bathroom

Partial SLEEC formalization is given below (and also available in Tutorial1/Amie.sleec)

  def_start

//events
event OpenDoor
event CloseDoor
event LockDoor
event UnlockDoor
event TurnOnLight
event TurnOffLight
event TurnMusicOn
event TurnMusicOff
event SwitchSong
event IncreaseVolume
event DecreaseVolume
event OpenCurtain
event CloseCurtain
event UserRequestLockDoor

//measures
measure doorOpened:**boolean**
measure doorLocked:**boolean**
measure curtainOpened:**boolean**
measure userAssent:**boolean**   *//user accepts an action*
measure userInDanger**: boolean**
measure soundlevel: **scale**(low, medium, loud)
measure weekday: **boolean**
measure timeDay: **scale** (morning, lunch, afternoon, evening, night)
measure lightLevel: **scale** (dark, dayLight, bright, extraBright)
measure userLocation: **scale** (kitchen, livingRoom, bedroom, bathroom, hallway)

def_end

// Rules
rule_start

 // Social rule: Ensure that shared spaces are accessible to all authorized individuals.
R1 **when** UserRequestLockDoor **and** (({userLocation} = kitchen) **or** ({userLocation} = livingRoom)) **then not** LockDoor **unless** {userInDanger}

// Legal rule: Adhere to local laws regulating acceptable sound levels.
R2 **when** TurnMusicOn **and** ({soundlevel} = loud)
   then DecreaseVolume **within** 30 **seconds unless** (not {weekday})

// Ethical rule: Use energy-efficient lighting solutions where possible.
R3 **when** TurnOnLight **and** ({lightLevel} = bright)
    then TurnOffLight **within** 20 **seconds unless** (**not** {userAssent})

// [COMPLETE HERE - Task 1.a]

rule_end

concern_start
    // When music is turned on and the sound level is already loud, increasing the volume to make it excessively loud and potentially harmful.
    c1 **when** TurnMusicOn **and** ({soundlevel} = loud) **then** IncreaseVolume
    //[Complete here - Task 1.b]
concern_end

purpose_start
    // Ensuring that the user can lock the door
    p1 **exists** UserRequestLockDoor and (not {doorLocked})
    //[Complete here - Task 1.c]
purpose_end

Task 1.a: *Complete the rules above in the **Amie.sleec**, by extending or updating the SLEEC definitions and rules (e.g., by adding a defeater), and formalizing the following social, legal, and ethical rules in SLEEC (see Slides 17-22 in Part 1):*

- Social rule: Ensure privacy when a user is dressing
- Legal rule: For the locking and unlocking of the doors, ensure compliance with local regulations regarding fire exits and safety protocols.
- Ethical rule: Avoid playing extravagant content in shared spaces.

Task 1.b: **Amie.sleec** *has one example of a SLEEC concern, i.e., a bad behavior that the Amie system should avoid (see Slide 51 in Part 1). Define 1-3 other concerns.*

Task 1.c: **Amie.sleec** *has one example of a SLEEC purpose, i.e., a main system functionality that must not be constrained since otherwise the Amie system would not be useful (see Slide 54 in Part 1). Define 1-3 other purposes.*

Task 1.d: *Copy the completed SLEEC Amie specification into a file called <amie-your_name.sleec> and email it to* chechik@cs.toronto.edu *with the subject "Tutorial 1 Task 1 answer".*

Part 2: Extend SLEEC

We would like to extend SLEEC to add the following new operators between events (E1 and E2) and measures (m1):

- E1 **isContradictoryWith** E2 – impossible to have E1 and E2 occurring at the same time
- E1 **happenBefore** E2   – the event E1 always happen before E2
- **when** E1 **then** m1 **until** E2   – a measure assignment m2 is the consequence of the occurrence of an initial event E1, which persists until a subsequent event E2 occurs

Recall the semantics of the existing SLEEC operators over a trace (from Slide 25 in Part 1):

$\sigma \models_i p$	iff	$\mathbb{M}_i(p)$
$\sigma \models_i e$ within t	iff	$\exists j \in [i,n].(e \in \mathcal{E}_j \wedge \delta_j \in [\delta_i, \delta_i + \mathbb{M}_j(t)])$
$\sigma \not\models_i^j e$ within t	iff	$\delta_j = \delta_i + \mathbb{M}_i(t) \wedge \forall j' \in [i,j](e \notin \mathcal{E}_{j'})$
$\sigma \models_i$ not e within t	iff	$\exists j(\sigma \not\models_i^j e$ within $t)$
$\sigma \not\models_i^j$ not e within t	iff	$\sigma \models e$ within $t \wedge \forall j' \in [i,j)(\sigma \not\models_i^j e$ within $t)$
$\sigma \models_i (p \Rightarrow ob)$	iff	$\sigma \models_i p \Rightarrow \sigma \models_i ob$
$\sigma \not\models_i^j (p \Rightarrow ob)$	iff	$\sigma \models_i$ not $p \wedge \sigma \not\models_i^j ob$
$\sigma \models_i cob^+$ otherwise $\bigvee_{cob}$	iff	$\sigma \models_i cob^+ \vee \exists j(\sigma \not\models_i^j cob^+ \wedge \sigma \models_j \bigvee_{cob})$
$\sigma \not\models_i^j cob^+$ otherwise $\bigvee_{cob}$	iff	$\exists j' \in [i,j](\sigma \not\models_i^{j'} cob^+ \wedge \sigma \not\models_{j'}^j \bigvee_{cob})$
$\sigma \models$ when e and p then $\bigvee_{cob}$	iff	$\forall i \in [1,n]((e \in \mathcal{E}_i \wedge \mathbb{M}_i(p)) \Rightarrow \sigma \models_i \bigvee_{cob})$
$\sigma \models_i$ not $\bigvee_{cob}$	iff	$\exists j(\sigma \not\models_i^j \bigvee_{cob})$
$\sigma \models$ exists e and p while $\bigvee_{cob}$	iff	$\exists i \in [1,n](e \in \mathcal{E}_i \wedge \mathbb{M}_i(p) \wedge \sigma \models_i \bigvee_{cob})$
$\sigma \models$ exists e and p while not $\bigvee_{cob}$	iff	$\exists i \in [1,n](e \in \mathcal{E}_i \wedge \mathbb{M}_i(p) \wedge \sigma \models_i$ not $\bigvee_{cob})$

Semantics of normalized SLEEC DSL defined over trace $\sigma = (\mathcal{E}_1, \mathbb{M}_1, \delta_1) \ldots (\mathcal{E}_n, \mathbb{M}_n, \delta_n)$.

Task 2.a: *Select from the three possible choices below the the right semantic for the new operator `isContradictoryWith' and justify why:*

☐ $\sigma \models e_a$ **isContradictoryWith** e_b iff $\forall(\mathcal{E}_i, \mathbb{M}_i, \delta_i) \in \sigma \cdot (e_a \in \mathcal{E}_i \wedge \neg e_b \in \mathcal{E}_i)$

☐ $\sigma \models e_a$ **isContradictoryWith** e_b iff $\forall(\mathcal{E}_i, \mathbb{M}_i, \delta_i) \in \sigma \cdot \neg(e_a \in \mathcal{E}_i \wedge e_b \in \mathcal{E}_i)$

☐ $\sigma \models e_a$ **isContradictoryWith** e_b iff $\forall(\mathcal{E}_i, \mathbb{M}_i, \delta_i) \in \sigma \cdot (e_a \in \mathcal{E}_i \vee e_b \in \mathcal{E}_i)$

Justification:

Task 2.b: *Select from the three possible choices below the the right semantics for the new operator `E1 **happenBefore** E2'and justify why:*

☐ $\quad \sigma \vDash e_a \text{ \textbf{happenBefore}} \ e_b \qquad$ iff $\quad \exists (\mathcal{E}_i, \mathbb{M}_i, \delta_i) \in \sigma \cdot (e_a \in \mathcal{E}_i \wedge \forall (\mathcal{E}_j, \mathbb{M}_j, \delta_j) \in \sigma \cdot (e_b \in \mathcal{E}_j \Rightarrow i > j))$

☐ $\quad \sigma \vDash e_a \text{ \textbf{happenBefore}} \ e_b \qquad$ iff $\quad \exists (\mathcal{E}_i, \mathbb{M}_i, \delta_i) \in \sigma \cdot (e_a \in \mathcal{E}_i \wedge \exists (\breve{\mathcal{E}}_j, \breve{\mathbb{M}}_j, \breve{\delta}_j) \in \sigma \cdot (e_b \in \breve{\mathcal{E}}_j \Rightarrow i > j))$

☐ $\quad \sigma \vDash e_a \text{ \textbf{happenBefore}} \ e_b \qquad$ iff $\quad \exists (\mathcal{E}_i, \mathbb{M}_i, \delta_i) \in \sigma \cdot (e_a \in \mathcal{E}_i \wedge \forall (\breve{\mathcal{E}}_j, \breve{\mathbb{M}}_j, \breve{\delta}_j) \in \sigma \cdot (e_b \in \breve{\mathcal{E}}_j \Rightarrow i >= j))$

Justification:

Task 2.c: *Select from the three possible choices below the the right semantics for the new operator `until' and add the justification for your choice:*

☐ $\quad \sigma \vDash \text{\textbf{when}} \ e_a \text{ \textbf{then}} \ p_b \text{ \textbf{until}} \ e_c \qquad$ iff $\quad$
$$\forall (\mathcal{E}_i, \mathbb{M}_i, \delta_i) \cdot e_a \in \mathcal{E}_i \Rightarrow (\exists (\mathcal{E}_j, \mathbb{M}_j, \delta_j \cdot e_b \in \mathcal{E}_j \wedge \forall k \in [i,j) \cdot \mathbb{M}_j(p_b) = \top)$$
$$\vee \forall (\mathcal{E}_j, \mathbb{M}_j, \delta_j) \in \sigma \cdot j \geq i \Rightarrow \mathbb{M}_j(p_b) = \bot)$$

☐ $\quad \sigma \vDash \text{\textbf{when}} \ e_a \text{ \textbf{then}} \ p_b \text{ \textbf{until}} \ e_c \qquad$ iff $\quad$
$$\forall (\mathcal{E}_i, \mathbb{M}_i, \delta_i) \cdot e_a \in \mathcal{E}_i \Rightarrow (\exists (\mathcal{E}_j, \mathbb{M}_j, \delta_j \cdot e_b \in \mathcal{E}_j \wedge \forall k \in [i,j) \cdot \mathbb{M}_j(p_b) = \bot)$$
$$\vee \forall (\mathcal{E}_j, \mathbb{M}_j, \delta_j) \in \sigma \cdot j \geq i \Rightarrow \mathbb{M}_j(p_b) = \top)$$

☐ $\quad \sigma \vDash \text{\textbf{when}} \ e_a \text{ \textbf{then}} \ p_b \text{ \textbf{until}} \ e_c \qquad$ iff $\quad$
$$\forall (\mathcal{E}_i, \mathbb{M}_i, \delta_i) \cdot e_a \in \mathcal{E}_i \Rightarrow (\forall (\mathcal{E}_j, \mathbb{M}_j, \delta_j \cdot e_b \in \mathcal{E}_j \wedge \forall k \in [i,j) \cdot \mathbb{M}_j(p_b) = \top)$$
$$\vee \exists (\mathcal{E}_j, \mathbb{M}_j, \delta_j) \in \sigma \cdot j \geq i \Rightarrow \mathbb{M}_j(p_b) = \bot)$$

Justification:

Part 3: Analysis of SLEEC rules well-formedness

*You will manually and automatically analyze the well-formedness of SLEEC rules for an automatic vacuum and mopping cleaner called Rumba. The SLEEC rules are provided below and also available in the file **Tutorial1/Rumba.sleec**.*

SLEEC rules for a Rumba

```
// Event and measure definitions
def_start
  // events
  event StartVacuum
  event EndVacuum
  event MemorizeMap
  event AvoidObstacles
  event EmptyGarbage
  event Moves
  event TurnedOn
  event TurnedOff
  event ClimbCarpet
  event StartMoping
  event DetectStairs
  event MoveBackward
```

event ChargeBattery
        event RequestMoping
        event RequestVacuum

        // measures
        measure turnOn: **boolean**
        measure garbageFull: **boolean**
        measure mapComplete: **boolean**
        measure batteryLevel: **numeric**
        measure mopingRequested: **boolean**
        measure vacuumingRequested: **boolean**
    def_end

// Rules
rule_start
    R1 when TurnedOn **then** StartVacuum **within** 1 **minutes** **unless** {garbageFull} **then** EmptyGarbage

    R2 **when** StartVacuum **then** MemorizeMap **within** 20 **minutes unless** {mapComplete}

    R3 **when** StartVacuum **then** AvoidObstacles

    R4 **when** StartVacuum **then not** StartMoping

    R5 **when** EndVacuum **then** StartMoping

    R6 **when** DetectStairs **then** MoveBackward

    R7 **when** ClimbCarpet **then** StartVacuum **within** 2 **minutes**

    R8 **when** StartMoping **then** AvoidObstacles

    R9 **when** EndVacuum **then** EmptyGarbage **unless** ({batteryLevel} < 20) **then** ChargeBattery

    R10 **when** TurnedOn **and** ({batteryLevel} < 20) **then** ChargeBattery

    R11 **when** StartMoping **then not** StartVacuum

    R12 **when** RequestMoping **then** StartMoping **unless** {garbageFull}

    R13 **when** RequestVacuum **then** StartVacuum **unless** ({batteryLevel} < 20) **then** ChargeBattery

    R14 **when** EndVacuum **then** TurnedOff
rule_end

Manual analysis:

Looking at the rules, answer the following questions:

Task 3.a: *Are Rumba rules R4 and R1 redundant?*
Task 3.b: *Are Rumba rules R5 and R14 vacuously conflicting?*
Task 3.c: *Are Rumba rules R4, R5, R6, R11, R12, and R3 situational conflicting?*

Automated analysis:

Use LEGOS-SLEEC to analyze and resolve the well-formedness of the above SLEEC rules. To do so:

1. If LEGOS-SLEEC is not installed for you, please follow the instructions in the README file to install in.
2. Launch LEGOS-SLEEC using: `python3 sleecFrontEnd.py`
   (sometimes you might need to do `python sleecFrontEnd.py`)
3. Copy the Rumba definitions and rules above to the LEGOS-SLEEC IDE
4. **Task 3.d:** *Check for vacuous conflicts by clicking on* `check conflict`*. If any conflicts are present, modify the rules to resolve the issue and check again to ensure the conflicts are resolved. Continue this process until there are no vacuous conflicting rules remaining.*
5. **Task 3.e:** *Check for situational conflicts by clicking on* `check situational conflict`*. If any conflicts are present, modify the rules to resolve the issue and check again to ensure the conflicts are resolved. Continue this process until there are no situational or vacuous conflicting rules.*
6. **Task 3.f:** *Check for redundant rules by clicking on* `check redundancy`*. If any redundant rules are present, modify the rules to resolve the issue and check again to ensure the redundancies are resolved. Continue this process until there are no redundant, situational, or vacuous conflicting rules.*
7. **Task 3.g:** *Copy the fixed SLEEC Rumba specification into a file called <rumba-your_name.sleec> and email it to chechik@cs.toronto.edu with the subject "Tutorial 1 Task 3 answer". You do not need to send your answers to 3a, 3b or 3c.*