Homework #8

Homework #1

Task #5

Подогревает воду до 3 разных температур и поддерживает заданную температуру. Крышка на замке, пока требуемая температура не достигнута. Защита от отсутствия воды.

Термопот

- Термопот предназначен для нагрева воды до 3 разных температур и поддержания данных температур
- Имеет крышку на замке, которая блокируется на время нагрева воды
- Наличие воды фиксируется датчиком
- Температура воды фиксируется датчиком
- Имеются 3 кнопки для выбора температуры воды [которые также “запускают” термопот]
- Имеется кнопка для включения режима поддержания температуры

Требования и процессы

- **Параллельные процессы**
  1. 3 кнопки выбора температуры
  2. Кнопка поддержания температуры
  3. Датчик температуры воды
  4. Датчик наличия воды
  5. Замок на крышке
  6. Обработчик сигналов датчиков (кнопок и регистрирующих температуру)
  7. Нагреватель воды
  8. Температура воды
  9. Вода
  10. Пользователь
- **Требования к системе**
  1. При отсутствии воды нагрев не производится
  2. Для старта нагрева воды крышка должна быть закрыта

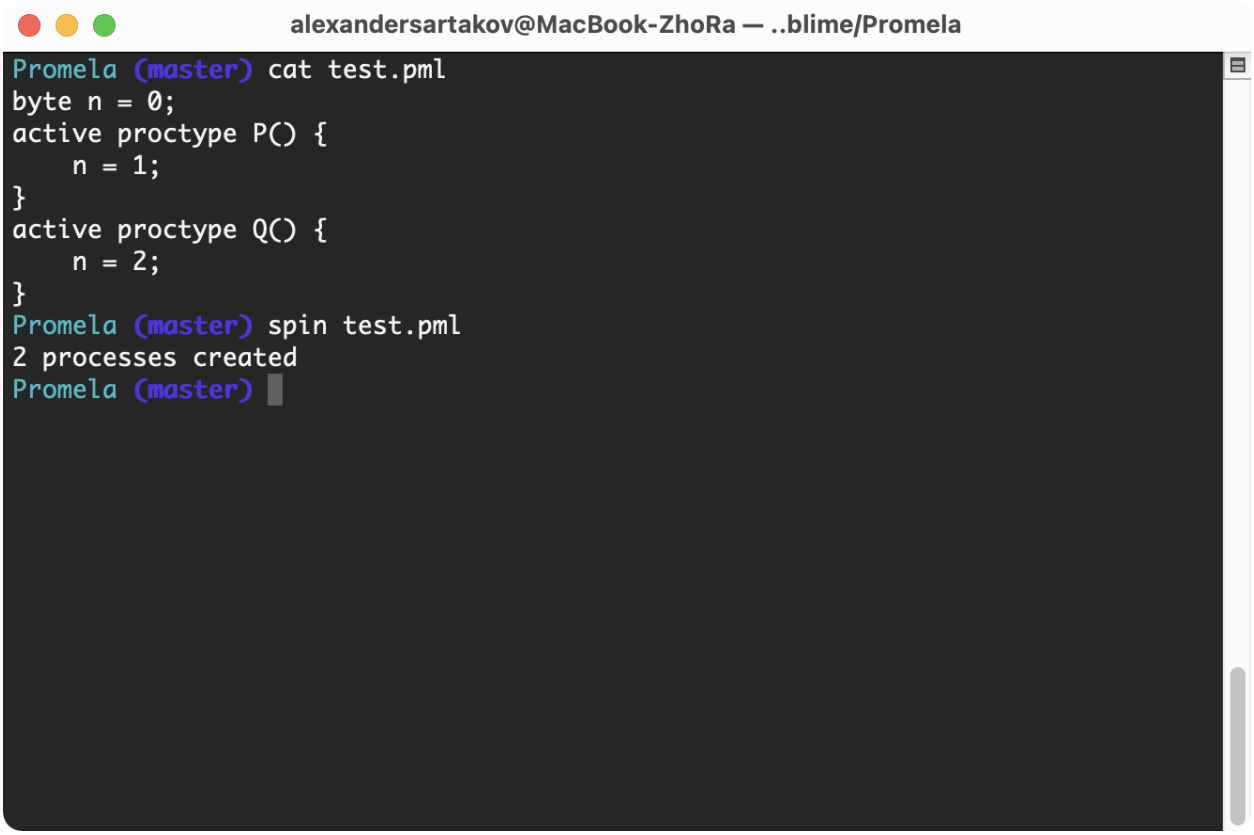
3. Нагрев начинается после нажатия на одну из кнопок при закрытой крышке и наличии воды
4. При нагревании или поддержании температуры крышка блокируется
5. При нажатии кнопки “поддержание температуры” термопот поддерживает последнюю заданную температуру

События-состояния процессы

- Пользователь
  - absent
  - pressing_1
  - pressing_2
  - pressing_3
  - pressing_temp
  - close_lid
  - open_lid
- Вода
  - exist
  - not_exist
- Нагреватель воды
  - turn_on
  - turn_off
- Температура воды
  - increasing
  - decreasing
  - equals_1, equals_2, equals_3
- Кнопки для выбора температуры
  - pressing_1, pressed_1, released_1
  - pressing_2, pressed_2, released_2

    - pressing_3, pressed_3, released_3
- Кнопка “поддержание температуры”
    - pressing_temp, pressed_temp, released_temp
- Датчик наличия воды
    - is_poured, not_poured
- Датчик температуры
    - is_reached, not_reached
- Крышка
    - is_clossed, is_opened
    - is_locked, is_unlocked
- Обработчик сигналов кнопок и датчиков, контроллер
    - Почти все перечисленные события всех процессов

Simple Promela code executed with SPIN



Homework #2

User

Тут происходит следующее:

- Пользователя нет
- Пользователь открывает крышку и наливает или сливает воду, затем закрывает крышку
- Пользователь нажимает одну из кнопок, после чего может делать всё, что захочет (поэтому много связей возникает)

[pressing_temp] - кнопка режима “поддержания температуры”, для удобства, пусть пользователь нажимает на кнопку несколько раз для активации режима поддержания температуры

```
graph TD
a --> a[absent] & b(pressing_1) & c[pressing_2] & d[pressing_3] & g[open_lid]
g --> h[add_water] & i[decrease_water]
g & h & i --> f[close_lid]
```

```
b & c & d --> g

f --> g & b & c & d

b --> c & d & a & b
c --> b & d & a & c
d --> b & c & a & d
```

Вода

Ну, может есть, а может нет… Вечный вопрос

```
graph TD
a[exit] --> b[not_exist]
b --> a
a --> a
b --> b
```

Нагреватель воды

Либо он работает, либо он не работает… Такой себе бинарник

```
graph TD
a[turn_on] --> b[turn_off]
b --> a
a --> a
b --> b
```

Температуры воды

Собственно, может увеличиваться, может уменьшаться и в это время становиться одной из 3 нужных температур (+ если включена функция поддержания температуры, будет переходить из equals → equals)

Как видно из схемы, temp_1 > temp_2 > temp_3 (то есть установлен порядок)

```
graph TD
cold --> equals_1 --> equals_2 --> equals_3
equals_1 --> cold
equals_3 --> equals_2 --> equals_1
equals_1 --> |Hold the temp 1| equals_1
equals_2 --> |Hold the temp 2| equals_2
equals_3 --> |Hold the temp 3| equals_3
```

Кнопки для выбора температуры и поддержания температуры

Чтобы было понятно:

> У кнопки есть цикл pressed → pressing → released
>
> Кнопок 4 и из каждого конечного состояния можно перейти в начальное другой кнопки
>
> Поэтому схема выглядит неприятно, по факту, тут 4 линейных структуры, у которых начало и концы соединены друг с другом

```
graph TD

a1[pressing_1] --> a2[pressed_1] --> a3[released_1]
b1[pressing_2] --> b2[pressed_2] --> b3[released_2]
c1[pressing_3] --> c2[pressed_3] --> c3[released_3]

a3 --> b1; b1 --> a3
a3 --> c1; c1 --> a3

b3 --> c1; c1 --> b3
b3 --> a1; a1 --> b3

c3 --> a1; a1 --> c3
c3 --> b1; b1 --> c3
```

Датчик наличия воды

```
graph TD
a[poured] --> b[not_poured]
b --> a
a --> a
b --> b
```

Датчик температуры

```
graph TD
a[equals_1]
b[equals_2]
c[equals_3]
d[not_reached]

d --> a & b & c
a & b & c --> d
```

Крышка

```
graph TD
a[is_clossed]
b[is_opened]
c[is_locked]
d[is_unlocked]

a --> c --> d
d --> b
b --> a
d --> c
```

Обработчик сигналов кнопок и датчиков [контроллер]

It’s gonna be a little complicated

Instead of 3 additional schemas of different temps added “any”

```
graph TD
not_exist --> not_poured
exist -->  not_poured
exist --> poured --> is_closed
is_closed --> pressed_1 & pressed_2 & pressed_3
pressed_1 & pressed_2 & pressed_3 --> is_locked --> equals_1 & equals_2 & equals_3 --> is_unlocked
is_unlocked --> is_opened --> exist
```

Homework #3

1. При отсутствии воды нагрев не производится

   G ¬ (turn_on && not_poured)

2. Для старта нагрева воды крышка должна быть закрыта

   G (¬(turn_on && (is_opened || is_unlocked))

3. Нагрев начинается после нажатия на одну из кнопок при закрытой крышке и наличии воды

   G (pressed_any && (poured && is_closed && is_locked) → turn_on)

   pressed_any = (pressed_1 || pressed_2 || pressed_3)

4. При нагревании или поддержании температуры крышка блокируется

   G (turn_on → **F** is_locked)

5. Крышка остается закрытой до окончания нагрева

   G (is_locked → is_locked **W** turn_off)

6. При нажатии кнопки “поддержание температуры” термопот поддерживает заданную температуру

   G (pressed_temp → **F** equals_some)

   equals_some = equals_1 || equals_2 || equals_3

   pressed_temp = (

   pressed_1 → **F** pressed_1 ||

   pressed_2 → **F** pressed_2 ||

   pressed_3 → **F** pressed_3) # need to push 2 times in a row

Homework #4

Water

- aWater = {exist, not_exist}
- WaterInit = exist → Exist
- Exist = exist → Exist | not_exist → NotExist
- NotExist = not_exist → NotExist | exist → Exist

Water presence sensor

- aPresence = {poured, not_poured}
- PresenceInit = not_poured → NotPour
- NotPour = not_poured → NotPoured | poured → Pour
- Pour = poured → Pour

Water heater

- aHeater = {turn_on, turn_off}
- HeaterInit = turn_off → HeaterOff

- HeaterOff = turn_off → HeaterOff | turn_on → HeaterOn
- HeaterOn = turn_on → HeaterOn | turn_off → HeaterOff

Water temperature

- aTemp = {cold, equals_1, equals_2, equals_3}
- TempInit = cold → TempCold
- TempCold = cold → TempCold | equals_1 → cold → TempCold | cold → equals_1 → TempOne
- TempOne = equals_1 → TempOne | equals_2 → equals_1 → TempOne | equals_1 → equals_2 → TempTwo
- TempTwo = equals_2 → TempTwo | equals_3 → equals_2 → TempTwo | equals_2 → equals_3 → TempThree
- TempThree = equals_3 → TempThree | equals_3 → equals_2 → TempTwo

Water temperature sensor

- aSensor = {not_reached, equals_1, equals_2, equals_3}
- SensorInit = not_reached → SensorLow
- SensorLow = not_reached → SensorLow | equals_1 → SensorOne
- SensorOne = equals_1 → SensorOne | not_reached → SensorLow | equals_2 → SensorTwo
- SensorTwo = equals_2 → SensorTwo | equals_1 → SensorOne | equals_3 → SensorThree
- SensorThree = equals_3 → SensorThree | equals_2 → SensorTwo

Water temperature choosing buttons

There’s gonna 4 the same buttons, let me provide one to ease the checking process

- aButtons = {pressing, pressed, released}
- ButtonInit = released → Button

- Button = released → Button | pressing → pressed → released → Button

Lid

- aLid = {is_closed, is_locked, is_unlocked, is_opened}
- LidInit = is_opened → LidOpen
- LidOpen = is_opened → LidOpen | is_closed → LidClosed
- LidClosed = is_closed → LidClosed | is_opened → LidOpen | is_locked → LidLocked
- LidLocked = is_locked → LidLocked | is_unlocked → LidClosed

Controller

- aController = {not_exist, not_poured, exist, poured, is_closed, pressed_1, pressed_2, pressed_3, is_locked, equals_1, equals_2, equals_3, is_unlocked, is_opened, wait}
- ContInit = wait → Pressed
- Pressed = wait → Pressed | pressed_n→ (poured → is_closed → is_locked)→ WorkN
- WorkN = turn_on → WorkN | equals_n → turn_off → (is_unlocked → is_opened) → Open
- Opened = open → Opened

  where equals_n is shortened equivalent for 3 notes

User

- aUser = {absent, pressing_1, pressing_2, pressing_3, open_lid, add_water, decrease_water, close_lid}
- **non-deterministic behaviour**

Homework #5

1. Формализуйте 3 вида трейсов длины 10 Контроллера.
    a. C1 = <wait, pressed_1, poured, is_closed, is_locked, turn_on, equals_1, turn_off, is_unlocked, is_opened>
    b. C2 = <wait, pressed_2, poured, is_closed, is_locked, turn_on, equals_2, turn_off, is_unlocked, is_opened>
    c. C3 = <wait, pressed_3, poured, is_closed, is_locked, turn_on, equals_3, turn_off, is_unlocked, is_opened>
2. Постройте сужение (↾) трейсов относительно событий другого процесса.
    a. C1 ↾ aLid = <is_closed, is_locked, is_unlocked, is_opened> # fixed
    b. C1 ↾ aWater = <>
    c. C1 ↾ aPressence = <poured>
    d. C1 ↾ aHeater = <turn_on, turn_off>
    e. C1 ↾ aTemp = <equals_1>
    f. C1 ↾ aSensor = <equals_1>
    g. C1 ↾ aButton = <pressed_1>
3. Сравните исходные трейсы и их сужения
    a. Они несравнимы, ибо стартовое состояние есть только у контроллера → его нельзя сравнить с его суждением
4. Выберите короткий процесс и формализуйте его трейс.
    a. ⋃ m≥0 ⋃ n≥0 ⋃k ≥ 0 { s | s ≤ (<turn_on>m ^ <turn_off>n) k } # fixed
5. Запишите процесс P after s, где P -- Контроллер, a s -- трейс длины 3.

    s = <wait, pressed_1, poured>

    P/s = is_closed → is_locked → WorkN (in this case, supposed to be WorkOne)
6. Определите, являются ли циклическими в CSP-смысле все процессы Вашей системы. Для одного из трейсов п.1 найдите кратчайший трейс, который приводит к исходному процессу: ∃ t : P / (c1^t) = P.

    C1 = <wait, pressed_1, poured, is_closed, is_locked, turn_on, equals_1, turn_off, is_unlocked, is_opened>

    t = <not_poured, wait>

Homework #6

1. Формализуйте спецификацию 3 свойств Контроллера в CSP-терминах.

   a. ThermopotSafety = ((tr ↓ equals) ≤ (tr ↓ poured)) #нагрев не происходит без воды

   b. NormalActivity = (tr ↓ poured)=(tr ↓ is_closed) -> (tr ↓ equals_n) ≤ (tr↓pressed_n)

   c. PerfectUser = ((tr ↓ is_opened) ≤ (tr ↓ is_unlocked)) #Пытается открыть крышку, когда она не под замком

2. Задайте параллельное исполнение двух процессов своей системы с пересекающимися алфавитами с помощью законов для параллельного исполнения опишите P1||P2 без использования ||. Должны быть приведены процессы P1, P2 и P1||P2.

```
P = (a -> P1 | b -> P2) и Q = (c -> Q1 | d -> Q2). c \in aP.
Тогда P || Q = (a -> P1||Q) |  (b -> P2||Q) | (d -> P||Q2).
```

   a. **Pressed** = wait → Pressed | pressed_n → (poured → is_closed → is_locked) → WorkN

      a. **Pressed** = pressed → poured → is_closed → is_locked → WorkN | wait → Pressed

   b. **LidClosed** = is_locked → LidLocked | is_opened → LidOpen

   c. *By definition*
      Pressed || LidClosed = (pressed → poured → is_closed → is_locked → WorkN | wait → Pressed) || (is_locked → LidLocked | is_opened → LidOpen)

      Applied L7: (c in aP) ⇒
      Pressed || LidClosed = (pressed → poured → is_closed → is_locked → WorkN || LidClosed) | (wait → Pressed || LidClosed) | (is_opened → Pressed || LidOpen) ⇒

      Removing "||" 3 times ⇒

Pressed || LidClosed = (pressed → poured → is_closed → ((is_locked → WorkN) | (is_locked → LidClosed))) | ((wait → Pressed) | (wait → LidClosed)) | ((is_opened → Pressed) | (is_opened → LidOpen)) ⇒

Removing unnecessary brackets ⇒
Pressed || LidClosed = pressed → poured → is_closed → ((is_locked → WorkN) | (is_locked → LidClosed)) | (wait → Pressed | wait → LidClosed) | (is_opened → Pressed | is_opened → LidOpen) ⇒

Separating the equations ⇒
Pressed || LidClosed = pressed → poured → is_closed → is_locked → WorkN | pressed → poured → is_closed → is_locked → LidClosed | wait → Pressed | wait → LidClosed | is_opened → Pressed | is_opened → LidOpen

Homework #7

1. Записать процесс BOOL со стр. 22 лекции 7.

DD = (setorange → O | setlemon → L)
O = (orange → O | setlemon → L | setorange → O)
L = (lemon → L | setorange → O | setlemon → L)

X5. The behaviour of a Boolean variable used by a computer program.
BOOL = f(DD)
f(setorange) = assign0
f(setlemon) = assign1
f(orange) = fetch0
f(lemon) = fetch1

BOOL = (assign0 → O | assign1→ L)
O = (fetch0→ O | assign1→ L | assign0→ O)
L = (fetch1→ O | assign0→ O | assign1→ L)

2. Определить процессы системы верхнего уровня, которые получаются изменением символов. Записать одну изменяющую функцию.

a. ButtonInit, LidInit

b. ButtonInit2 = f(ButtonInit3)
f(pressed_2) = pressed_3
f(pressing_2) = pressing_3
f(released_2) = released_3 () ⇒ can choose any of 3 buttons

3. Построить полное описание своей параллельной системы с использованием переименования и/или разметки процессов.
Отметить изменения в CSP-записях процессов (скорее всего, это контроллер) в результате переименований/разметки.

Termopot = ContInit || WaterInit || HeaterInit || LidInit || (a : buttonInit) || (b : waterTemperature) || User || PresenceInit || SensorInit

Controller

- aController = {not_exist, not_poured, exist, poured, is_closed, pressed_1, pressed_2, pressed_3, is_locked, equals_1, equals_2, equals_3, is_unlocked, is_opened, wait}
- ContInit = wait → Pressed
- Pressed = wait → Pressed | a.pressed_n→ (poured → is_closed → is_locked)→ WorkN
- WorkN = turn_on → WorkN | b.equals_n → turn_off → (is_unlocked → is_opened) → Open
- Opened = open → Opened | a.pressed_n → Pressed

  where equals_n is shortened equivalent for 3 notes

4. Записать модифицированный процесс FOOTMAN. Доказать, что этот процесс гарантирует сытость философов (+5).

The effective solution is to buy more forks and plenty of spaghetti.
To guarantee that a seated philosopher will eventually eat modify the behaviour of footman: Having helped a philosopher to his seat he waits until that philosopher has picked up both forks before he allows either of his neighbours to sit down

FOOT(j) - defines the behaviour of footman with j philosophers

$$FOOT_0 = (x : D \rightarrow FOOT_1)$$
$$FOOT_j = (x : D \rightarrow FOOT_{j+1} \mid y : U \rightarrow FOOT_{j-1}) \quad \text{for } j \in \{1, 2, 3\}$$
$$FOOT_4 = (y : U \rightarrow FOOT_3)$$

- $U = \bigcup_{i=0}^{4}\{i.gets\ up\}$ $D = \bigcup_{i=0}^{4}\{i.sits\ down\}$

Let's define **philosopher** and **fork** behaviour

Phil = sit → get_one_fork → get_second_fork → put_one_fork → put_second_fork → stand → **Phil**

Fork = left_up → left_down → Fork | right_up → right_down → **Fork**

Получаем, следующее поведение для лакея:

1. Лакей ждет философа
2. Лакей садит философа за стол и не дает сесть никому рядом с ним, пока философ не возьмет обе вилки в руки. (В это время лакей может посадить философа через место от философа, тогда конкуренция за вилки не случится)

Запишем поведение лакея:

Footman_init = wait → Footman

Footman = wait → Footman | sit_phil → wait_for_philospher_taking_forks_1 → take_first_fork → wait_for_philospher_taking_forks_2 → take_second_fork → Footman

Где wait_for_philosopher_taking_forks - означает, что лакей не садит рядом с философом никого, пока тот не возьмет обе вилки

Требуется доказать **сытость философов**, следовательно, нужно определить, что это такое.

1. Все философы могут сесть за стол
2. Каждый философ может дождаться своей очереди для того, чтобы поесть == не возникает deadlock’a

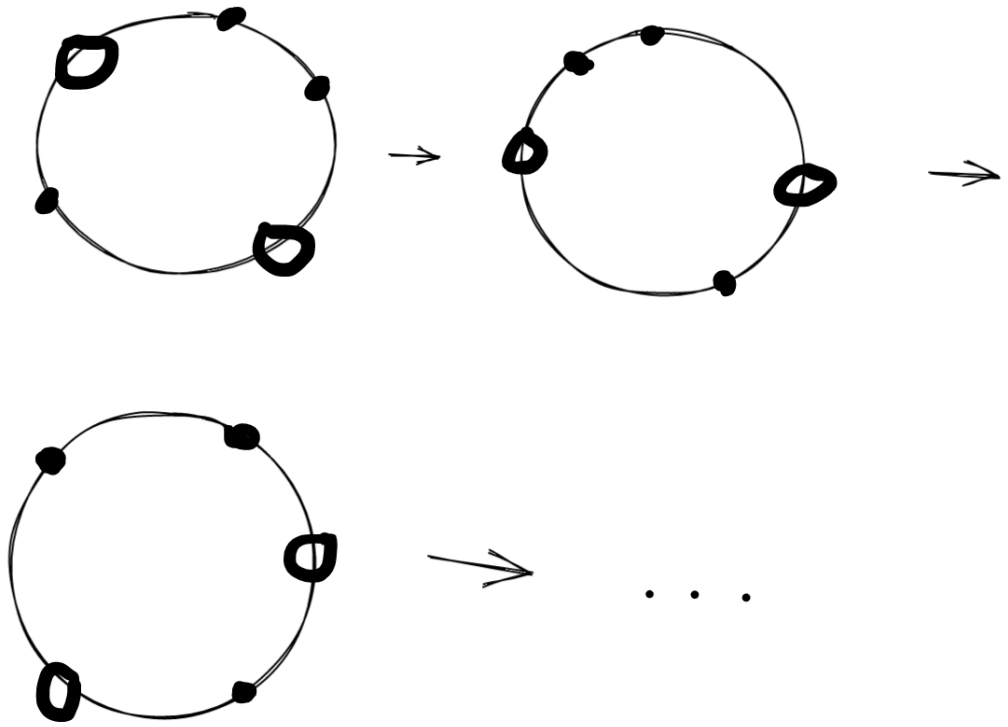
Доказательство:

1. Пока за столом никого нет, лакей может посадить философа на любое место, тогда философ может взять обе вилки в руки и не бороться за них.
2. Когда за столом сидит философ (и берет вилки или ест), лакей садит другого философа через место от философа, чтобы второй философ мог взять обе вилки и поесть (пока философ не возьмет вилки в руки рядом с ним нельзя садить другого)
3. Оставшиеся философы будут садится рядом с уже обедающими философами (потому что пока они не взяли обе вилки в руки, садить рядом с ними нельзя) и брать вилку как только она освобождается от соседнего философа (он уходит и освобождает 2 вилки). В это время (пока философ сидит с одной или вовсе без вилок в руках) рядом с ним по условию нельзя садить других ⇒ deadlock не

возникает (то есть нет варианта, когда все философы будут вечно сидеть и ждать вилки)
Так же это означает, что за стол можно посадить всех философов по очереди → первые два философа - очевидно из пункта 1 и 2, еще двух садим рядом с ними, оставшийся ждет. Когда один из философов поесть и встанет, сидящий рядом с ним получает вилку и начинает есть, тогда последнего стоящего философа можно посадить за стол и таким образом запустить цикл опять

Таким образом, получаем, что каждый философ сможет насытиться

Если как-то пытаться нарисовать, то получаем такую картинку:



Где большие круглишки - обедающие философы, которые меняются через итерацию (ну тут для наглядности представлено 2 итерации, то есть поело 2 философа (просто было трудно рисовать кружки))

Homework #8

1. Найти в своей системе процессы, параллельная комбинация которых точно не имеет тупиков (deadlocks).

1. **Contorller** имеет пересечение со всеми процессами системы мощностью > 1, кроме **User, Button** → deadlock не будет возникать с процессами User и Button, со всеми остальными deadlock может возникнуть
2. Паралельная комбинация всех остальных процессов *(кроме **Water Temperature** и **Water temperature sensor**, так как они имеют пересечение алфавитов, поэтому их параллельная комбинация может привести к deadlock)* не имеет deadlock'ов

2. Записать процесс пользователя (внешней среды), используя строго недетерминированный выбор.

- User:
  - aUser = {absent, pressing_1, pressing_2, pressing_3, open_lid, add_water, decrease_water, close_lid}
  - UserInit = absent → User
  - User_process = (absent → User) П
    (lid_process) П
    (pressing_1 → process) П
    (pressing_2 → process) П
    pressing_3 → process)

    process = (lid_process П (pressing_1 → process) П (pressing_2 → process) П (pressing_3 → process))

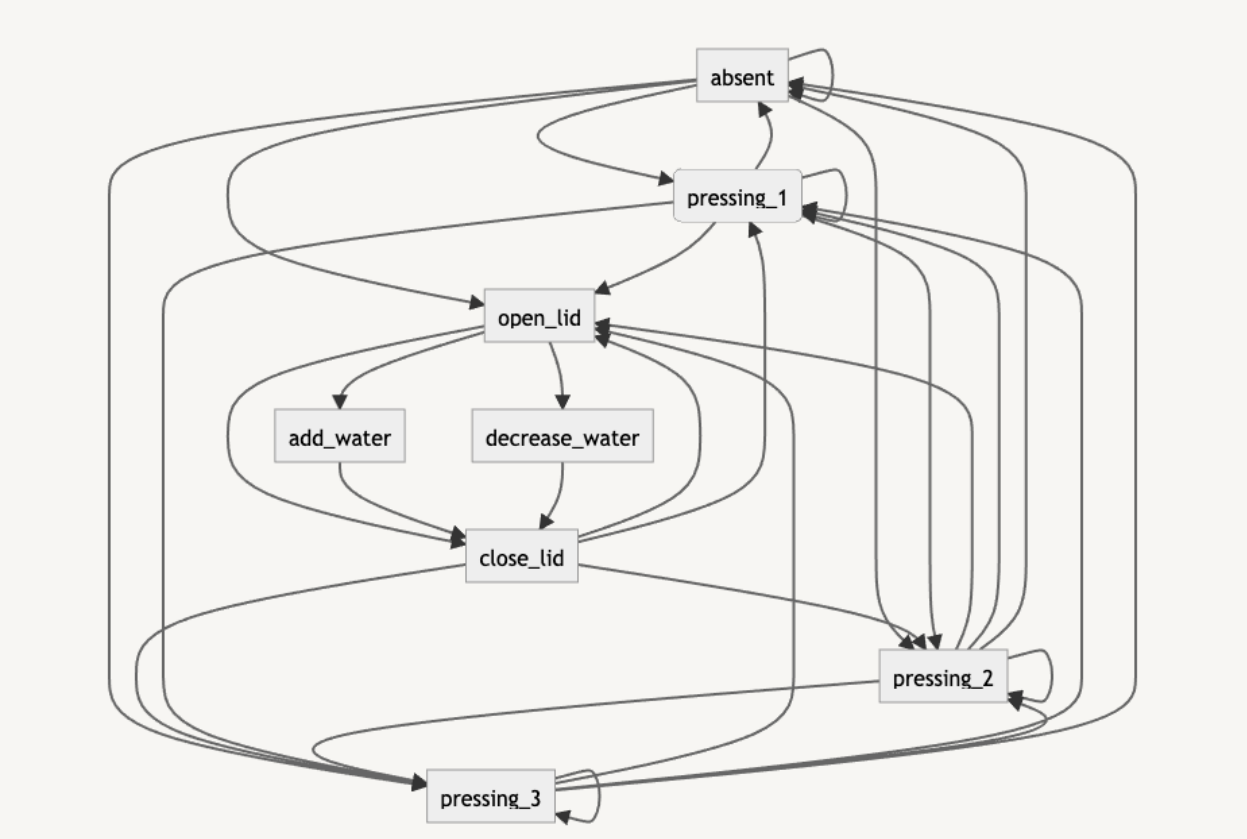
    // pressing_n перед process дает понять в каком состоянии находится система

    lid_process = open_lid → ((add_water → CloseLid) П (decrease_water → CloseLid) П CloseLid)

    CloseLid = close_lid → process

3. Записать спецификацию недетерминированного процесса пользователя (среды) с учётом отказов

CloseLid = (tr ↓ open_lid > tr ↓ close_lid ⇒ close_lid ∉ ref)


absent
pressing_1
open_lid
add_water
decrease_water
close_lid
pressing_2
pressing_3