Контрольна робота №2. Частина 2

Формальна верифікація обраної системи

«Турнікет»

Шептуха Євгеній, ШІ

1. Перегляд підходу до специфікації

Словесний опис обраної системи:

Турнікет обладнаний приймачем карт, механізмом для блокування/розблокування проходу, таймером і двома індикаторами: "Прохід дозволено" та "Прохід заборонено".

У початковому стані турнікет забороняє прохід: індикатор "Прохід заборонено" увімкнений, індикатор "Прохід дозволено" вимкнений, прохід заблокований, а таймер не активний.

Коли користувач прикладає проїзну картку, система перевіряє її дійсність: якщо картка ще чинна (не минув термін дії) і на ній залишилися поїздки, то одна поїздка списується, індикатор "Прохід дозволено" вмикається, "Прохід заборонено" вимикається, прохід розблоковується, і запускається таймер на 5 секунд.

Якщо ж картка недійсна або на ній немає поїздок, жодних змін у роботі турнікету не відбувається.

Після завершення п’яти секунд таймер зупиняється, індикатор "Прохід дозволено" вимикається, "Прохід заборонено" вмикається, прохід блокується, і система повертається до початкового стану очікування.

Початково для специфікації даної системи було обрано Z-нотацію. Було обрано саме цю мову специфікації, оскільки вона є доволі розповсюдженою, а також була одним з прикладів до самостійного виконання у підручнику Л.Л. Омельчук «Формальні методи специфікації програм».

При подальшому аналізі можливостей цієї специфікації було виявлено, що Z-нотація є непідходящою для специфікації та аналізу систем, що містять часовий та паралельний аспекти[1].

Тому для специфікації системи турнікету було (заново) обрано мову TLA+, оскільки вона найбільше пасує для аналізу паралельних дискретних систем.

2. Опис обраного методу специфікації

TLA+ - це формальна мова специфікацій, розроблена Леслі Лампортом у 1999 році для проектування, моделювання та верифікації програмних систем, особливо розподілених. Її основу становлять темпоральна логіка дій (TLA) і математична теорія множин, що забезпечує точність і однозначність специфікацій.

Основні концепції

1. Дії та інваріанти: Система моделюється через набори дій (змін станів), а інваріанти визначають незмінні умови її коректності.
2. Перевірка моделей: Інструмент TLC аналізує всі можливі стани системи в **обмеженому просторі**, виявляючи порушення властивостей безпеки та життєздатності.
3. Перевірка доведень: Мова підтримує формальні докази правильності алгоритмів у декларативному стилі.

PlusCal

Це алгоритмічна мова верхнього рівня, що транслюється в TLA+ (подібно до того, як C компілюється в асемблер). Вона спрощує опис алгоритмів завдяки:

- Структурі, близькій до псевдокоду (наприклад, цикли while, умовні оператори).
- Автоматичній генерації специфікацій для подальшої перевірки в TLC.

Переваги TLA+

- Точність: Специфікації уникають неоднозначностей природних мов.
- Раннє виявлення помилок: Моделювання допомагає знайти логічні вади до реалізації.
- Універсальність: Підходить для апаратних систем, протоколів, алгоритмів.

Недоліки

- Обмежена масштабованість: Перевірка моделей ефективна лише для систем із обмеженою кількістю станів.

3. Формальна специфікація обраної системи

Для зручності специфікації системи турнікету використовувалася імперативна мова (псевдокод) PlusCal, яка потім транслюється в pure TLA+ специфікацію за допомогою засобів TLA+ Toolbox.

Специфікація моделює два паралельні процеси: взаємодію користувача з турнікетом і роботу таймера, який контролює тривалість проходу.

Спочатку задаються необхідні для моделювання константи, які потім будуть використані при перевірці моделі (Model Checking) при верифікації:

PassTime: Тривалість дозволу на прохід після успішного прикладання картки.

MaxRidesNum: Максимальна кількість поїздок, які може мати картка.

UseTimes: Кількість взаємодій із системою, після яких її робота завершується (для обмеження простору для верифікації моделі).

Далі задаються змінні для задання специфікації:

accessAllowed: Визначає, чи дозволено прохід (індикатор «Прохід дозволено»).

accessDenied: Вказує, що прохід заборонено (індикатор «Прохід заборонено»).

locked: Чи заблоковано турнікет.

timer: Таймер, що контролює тривалість доступу.

cardValid: Чи є картка дійсною.

rides: Кількість поїздок на картці.

counter: Лічильник кількості оброблених користувачів (досягає UseTimes).

Далі, за допомогою блоку define задаються інваріанти на pure TLA+, що забезпечують узгодженість станів:

CardStillValid / RidesAvailable / AccessGranted: Логічні умови для надання доступу.

TimerExpired: Час доступу вичерпано — система має скинутися.

TypeInvariant: Гарантує правильні типи значень та допустимі діапазони.

TurnstileWorkEnd: Основний цикл завершується при counter >= UseTimes.

IndicatorsMutualExclusive: accessAllowed і accessDenied не можуть бути одночасно істинними або хибними.

LockFollowsAccess: Стан механізму блокування турнікета має відповідати дозволу на доступ.

TimerImpliesAccess: Якщо таймер працює, доступ ще активний.

TimerZeroImpliesInitialState: Якщо значення таймера дорівнює нулю, система перебуває в початковому (заблокованому) стані.

RidesNonNegative: Значення поїздок не може бути менше нуля.

ValidCardMeansRidesPositive: Дійсна картка повинна мати хоча б одну поїздку для надання доступу.

InvalidCardStateUnchanged: Якщо картка недійсна або не має поїздок, система залишається в початковому стані.

Далі задаються паралельні процеси на мові PlusCal:

timer_process (таймер)

Циклічно виконується та:

- Збільшує значення timer, якщо він активний.
- Скидає систему (блокує, забороняє доступ, обнуляє таймер), коли спливає PassTime.
- Очікує, поки таймер активний, доступ заборонено або досягнуто межу UseTimes.

user_process (користувач)

Імітує дії користувача, повторюється до UseTimes:

- Випадковим чином моделює одну з трьох дій:
    - **Дійсна картка** (із кількістю поїздок від 1 до MaxRidesNum)
    - **Недійсна картка** (протермінована)
    - **Дійсна картка без поїздок** (доступ має бути заборонений)
- Якщо картка дійсна і має поїздки, надається доступ:
    - Кількість поїздок зменшується, доступ дозволено, турнікет відкривається, запускається таймер.
- Очікує, поки турнікет знову заблокується, перед наступною дією.

Після написання коду на мові PlusCal, його було трансльовано у pure TLA+ мову за допомогою засобів TLA+ Toolbox.

Повний код специфікації на мові PlusCal та TLA+ для зручності наведено у Додатку А.

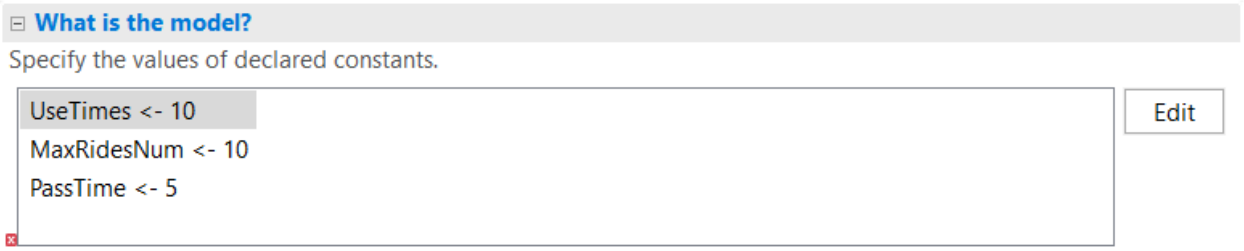
Перед розглядом наступного параграфу рекомендується ознайомитися із специфікаціями у додатках.

4. Формальна верифікація обраної системи

Для верифікації обраної системи використовувався засіб TLA+ Toolbox.

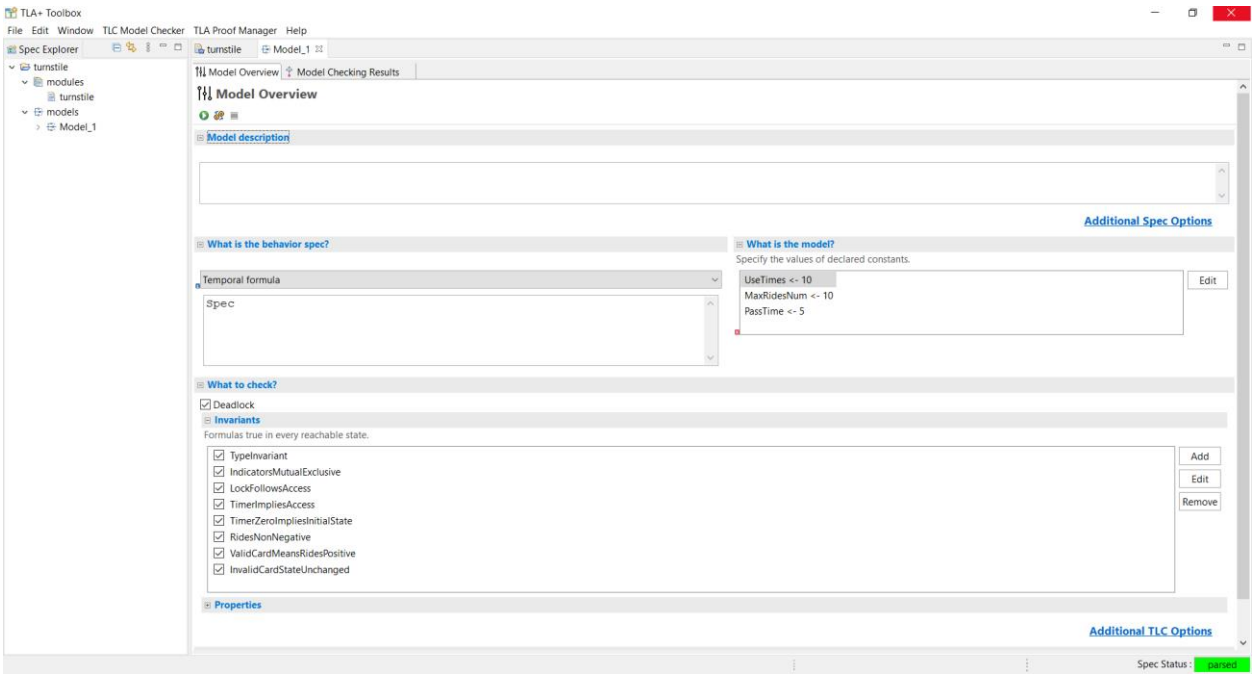
TLA+ Toolbox - це IDE (інтегроване середовище розробки) для інструментів TLA+. Воно може використовуватися для створення та редагування специфікацій, визначення синтаксичних помилок, трансляції PlusCal в pure TLA+, друку специфікацій у формат pdf, верифікації специфікацій за допомогою TLC Model Checker.

Для перевірки моделі за допомогою TLC Model Checker необхідно обмежити простір перебору станів. Цей простір було задано наступним чином:

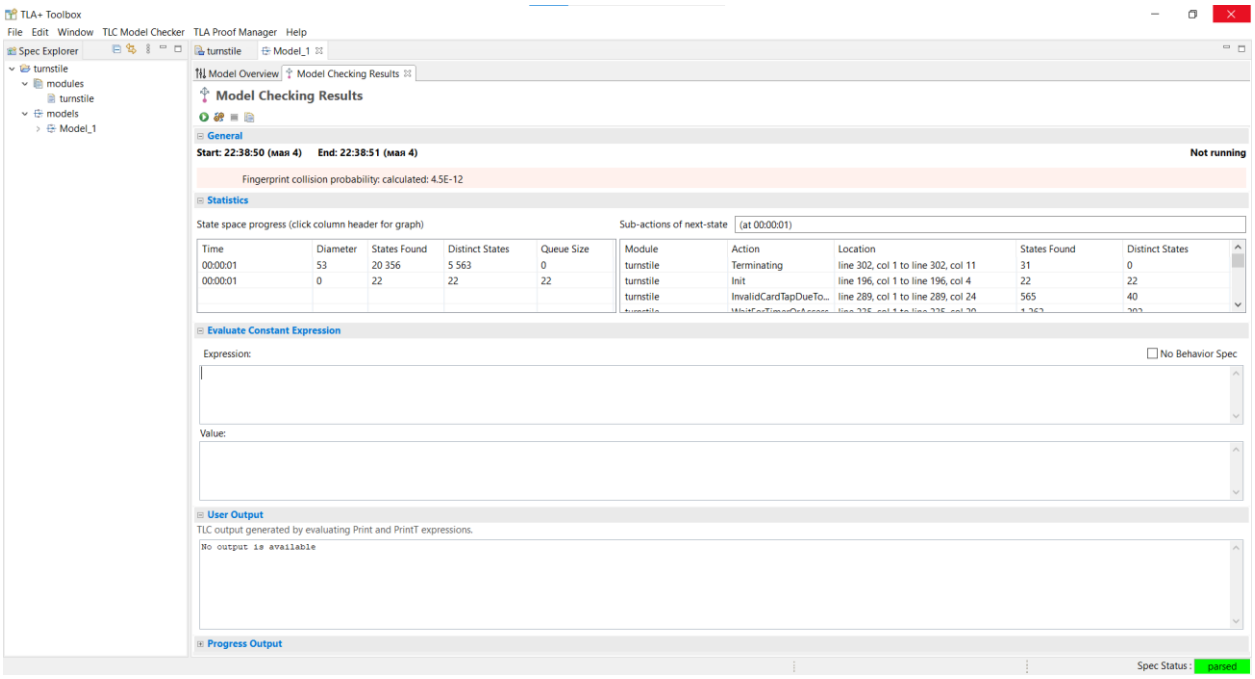


Також необхідно задати інваріанти системи для визначення відповідності моделі специфікації:

Вікно задання налаштувань верифікації має наступний виглад:



Після завершення верифікації (перевірки моделі) отримуємо наступні результати:



В даному вікні наведена інформація про коректність виконання (логи помилок), статистики по кількості перевірених станів, час виконання тощо.

Таким чином ми верифікували створену модель та переконалися у відовідності моделі до специфікації.

Література

[1] Mr. Vishal Ruhela, 2012, Z Formal Specification Language – An Overview, INTERNATIONAL JOURNAL OF ENGINEERING RESEARCH & TECHNOLOGY (IJERT) Volume 01, Issue 06 (August 2012)
[2] Омельчук Л.Л. Формальні методи специфікації програм. – К.: УкрІНТЕІ, 2010. - 78 с.
[3] https://learntla.com/index.html

[4] Specifying systems : the TLA+ language and tools for hardware and software engineers / Leslie Lamport.

Додаток А (див. нижче)

MODULE *turnstile*

EXTENDS *Integers*, *TLC*

CONSTANT
PassTime,
MaxRidesNum,
UseTimes

--algorithm *turnstile*
variables
accessAllowed = FALSE,
accessDenied = TRUE,
locked = TRUE,
timer = 0,
cardValid ∈ {TRUE, FALSE} ;
rides ∈ 0 .. *MaxRidesNum* ;
counter = 0 ;

define
CardStillValid ≜ *cardValid*
RidesAvailable ≜ *rides* > 0
AccessGranted ≜ *CardStillValid* ∧ *RidesAvailable*
TimerExpired ≜ *timer* ≥ *PassTime*

TypeInvariant ≜ ∧ *cardValid* ∈ BOOLEAN
∧ *accessAllowed* ∈ BOOLEAN
∧ *accessDenied* ∈ BOOLEAN
∧ *locked* ∈ BOOLEAN
∧ *timer* ∈ 0 .. *PassTime*
∧ *rides* ∈ 0 .. *MaxRidesNum*
∧ *counter* ∈ 0 .. *UseTimes*

TurstileWorkEnd ≜ *counter* ≥ *UseTimes*

The two indicators are never both active or both inactive
IndicatorsMutualExclusive ≜
(*accessAllowed* ∧ ¬*accessDenied*) ∨ (¬*accessAllowed* ∧ *accessDenied*)

The lock is only open (unlocked) when access is allowed
LockFollowsAccess ≜
locked = ¬*accessAllowed*

Timer is only non-zero if access is allowed (*i.e.*, during access window)
TimerImpliesAccess ≜
(*timer* > 0) ⇒ *accessAllowed*

When timer is zero, the system must be in the initial (locked) state
TimerZeroImpliesInitialState ≜

```
        (timer = 0) ⇒ ∧ accessAllowed = FALSE
                      ∧ accessDenied = TRUE
                      ∧ locked = TRUE

    Rides are never negative
    RidesNonNegative ≜
        rides ≥ 0

    A valid card always has at least one ride if ValidCard is true
    ValidCardMeansRidesPositive ≜
        cardValid = TRUE ⇒ (AccessGranted ≡ rides > 0)

    If the card is invalid or has no rides left, then the system remains in its initial/locked state
    InvalidCardStateUnchanged ≜
        (¬cardValid ∨ (rides = 0 ∧ timer = 0)) ⇒
            ∧ accessAllowed = FALSE
            ∧ accessDenied = TRUE
            ∧ locked = TRUE
            ∧ timer = 0
end define ;

process timer_process = "Timer"
begin
    TimerLoop:
        while (¬TurstileWorkEnd) do
            IncrementTimer:
                if timer > 0 then
                        timer := timer + 1 ;
                end if ;
            ResetSystem:
                if TimerExpired then
                        accessAllowed := FALSE ;
                        accessDenied := TRUE ;
                        locked := TRUE ;
                        timer := 0 ;
                end if ;
            WaitForTimerOrAccess:
                await timer > 0 ∨ ¬AccessGranted ∨ TurstileWorkEnd ;
        end while ;
end process ;

process user_process = "User"
begin
    UserActions:
        while (counter < UseTimes) do
            either
                Tap valid card with rides
```

```
                ValidCardTap:
                    with rides_num ∈ 1 .. MaxRidesNum do
                        cardValid := TRUE ;
                        rides := rides_num ;
                        counter := counter + 1 ;
                    end with ;
            or
                 Tap expired card
                InvalidCardTap:
                    with rides_num ∈ 1 .. MaxRidesNum do
                        cardValid := FALSE ;
                        rides := rides_num ;
                        counter := counter + 1 ;
                        skip    ;
                    end with ;
            or
                 Tap card with no rides
                InvalidCardTapDueToRides:
                    cardValid := TRUE ;
                    rides := 0 ;
                    counter := counter + 1 ;
                    skip ;
            end either ;

             Process card tap
            GrantAccess:
                if AccessGranted then
                        rides := rides − 1 ;
                        accessAllowed := TRUE ;
                        accessDenied := FALSE ;
                        locked := FALSE ;
                        timer := 1 ;
                end if ;

             Wait for system to reset before next action
            WaitForReset:
                await locked ∨ TurstileWorkEnd ;
        end while ;
end process ;
end algorithm   ;
 BEGIN TRANSLATION (chksum(pcal) = "bfa04102" ∧ chksum(tla) = "4a327817")
VARIABLES accessAllowed, accessDenied, locked, timer, cardValid, rides,
          counter, pc

 define statement
CardStillValid  ≜  cardValid
```

$RidesAvailable \triangleq rides > 0$
$AccessGranted \triangleq CardStillValid \land RidesAvailable$
$TimerExpired \triangleq timer \geq PassTime$

$$
\begin{aligned}
TypeInvariant \triangleq\ & \land cardValid \in \text{BOOLEAN} \\
& \land accessAllowed \in \text{BOOLEAN} \\
& \land accessDenied \in \text{BOOLEAN} \\
& \land locked \in \text{BOOLEAN} \\
& \land timer \in 0 \,..\, PassTime \\
& \land rides \in 0 \,..\, MaxRidesNum \\
& \land counter \in 0 \,..\, UseTimes
\end{aligned}
$$

$TurstileWorkEnd \triangleq counter \geq UseTimes$

$IndicatorsMutualExclusive \triangleq$
$\quad (accessAllowed \land \neg accessDenied) \lor (\neg accessAllowed \land accessDenied)$

$LockFollowsAccess \triangleq$
$\quad locked = \neg accessAllowed$

$TimerImpliesAccess \triangleq$
$\quad (timer > 0) \Rightarrow accessAllowed$

$TimerZeroImpliesInitialState \triangleq$
$\quad (timer = 0) \Rightarrow \land accessAllowed = \text{FALSE}$
$\qquad\qquad\qquad\quad \land accessDenied = \text{TRUE}$
$\qquad\qquad\qquad\quad \land locked = \text{TRUE}$

$RidesNonNegative \triangleq$
$\quad rides \geq 0$

$ValidCardMeansRidesPositive \triangleq$
$\quad cardValid = \text{TRUE} \Rightarrow (AccessGranted \equiv rides > 0)$

$InvalidCardStateUnchanged \triangleq$
$\quad (\neg cardValid \lor (rides = 0 \land timer = 0)) \Rightarrow$
$\qquad \land accessAllowed = \text{FALSE}$
$\qquad \land accessDenied = \text{TRUE}$
$\qquad \land locked = \text{TRUE}$
$\qquad \land timer = 0$

$vars \triangleq \langle accessAllowed,\ accessDenied,\ locked,\ timer,\ cardValid,\ rides,$
$\qquad counter,\ pc \rangle$

$$ProcSet \triangleq \{\text{“Timer”}\} \cup \{\text{“User”}\}$$

$$\begin{aligned}
Init \triangleq\ & \text{Global variables} \\
& \land accessAllowed = \text{FALSE} \\
& \land accessDenied = \text{TRUE} \\
& \land locked = \text{TRUE} \\
& \land timer = 0 \\
& \land cardValid \in \{\text{TRUE, FALSE}\} \\
& \land rides \in 0 \,..\, MaxRidesNum \\
& \land counter = 0 \\
& \land pc = [self \in ProcSet \mapsto \text{CASE } self = \text{“Timer”} \to \text{“TimerLoop”} \\
& \qquad\qquad\qquad\qquad \square\ \ self = \text{“User”} \to \text{“UserActions”}]
\end{aligned}$$

$$\begin{aligned}
TimerLoop \triangleq\ & \land pc[\text{“Timer”}] = \text{“TimerLoop”} \\
& \land \text{IF } (\neg TurstileWorkEnd) \\
& \qquad \text{THEN } \land pc' = [pc \text{ EXCEPT } ![\text{“Timer”}] = \text{“IncrementTimer”}] \\
& \qquad \text{ELSE } \land pc' = [pc \text{ EXCEPT } ![\text{“Timer”}] = \text{“Done”}] \\
& \land \text{UNCHANGED } \langle accessAllowed,\ accessDenied,\ locked,\ timer, \\
& \qquad\qquad cardValid,\ rides,\ counter\rangle
\end{aligned}$$

$$\begin{aligned}
IncrementTimer \triangleq\ & \land pc[\text{“Timer”}] = \text{“IncrementTimer”} \\
& \land \text{IF } timer > 0 \\
& \qquad \text{THEN } \land timer' = timer + 1 \\
& \qquad \text{ELSE } \land \text{TRUE} \\
& \qquad\qquad\ \ \land timer' = timer \\
& \land pc' = [pc \text{ EXCEPT } ![\text{“Timer”}] = \text{“ResetSystem”}] \\
& \land \text{UNCHANGED } \langle accessAllowed,\ accessDenied,\ locked, \\
& \qquad\qquad cardValid,\ rides,\ counter\rangle
\end{aligned}$$

$$\begin{aligned}
ResetSystem \triangleq\ & \land pc[\text{“Timer”}] = \text{“ResetSystem”} \\
& \land \text{IF } TimerExpired \\
& \qquad \text{THEN } \land accessAllowed' = \text{FALSE} \\
& \qquad\qquad\ \ \land accessDenied' = \text{TRUE} \\
& \qquad\qquad\ \ \land locked' = \text{TRUE} \\
& \qquad\qquad\ \ \land timer' = 0 \\
& \qquad \text{ELSE } \land \text{TRUE} \\
& \qquad\qquad\ \ \land \text{UNCHANGED } \langle accessAllowed,\ accessDenied,\ locked, \\
& \qquad\qquad\qquad\qquad timer\rangle \\
& \land pc' = [pc \text{ EXCEPT } ![\text{“Timer”}] = \text{“WaitForTimerOrAccess”}] \\
& \land \text{UNCHANGED } \langle cardValid,\ rides,\ counter\rangle
\end{aligned}$$

$$\begin{aligned}
WaitForTimerOrAccess \triangleq\ & \land pc[\text{“Timer”}] = \text{“WaitForTimerOrAccess”} \\
& \land timer > 0 \lor \neg AccessGranted \lor TurstileWorkEnd \\
& \land pc' = [pc \text{ EXCEPT } ![\text{“Timer”}] = \text{“TimerLoop”}] \\
& \land \text{UNCHANGED } \langle accessAllowed,\ accessDenied,\ locked, \\
& \qquad\qquad timer,\ cardValid,\ rides,\ counter\rangle
\end{aligned}$$

$$
\begin{array}{l}
\mathit{timer\_process} \triangleq \mathit{TimerLoop} \lor \mathit{IncrementTimer} \lor \mathit{ResetSystem} \\
\qquad\qquad \lor \mathit{WaitForTimerOrAccess}
\end{array}
$$

$$
\begin{array}{l}
\mathit{UserActions} \triangleq \land\ \mathit{pc}[\text{“User”}] = \text{“UserActions”} \\
\qquad\qquad \land\ \text{IF } (\mathit{counter} < \mathit{UseTimes}) \\
\qquad\qquad\qquad \text{THEN } \land\ \lor\ \land\ \mathit{pc}' = [\mathit{pc} \text{ EXCEPT } ![\text{“User”}] = \text{“ValidCardTap”}] \\
\qquad\qquad\qquad\qquad\qquad\ \lor\ \land\ \mathit{pc}' = [\mathit{pc} \text{ EXCEPT } ![\text{“User”}] = \text{“InvalidCardTap”}] \\
\qquad\qquad\qquad\qquad\qquad\ \lor\ \land\ \mathit{pc}' = [\mathit{pc} \text{ EXCEPT } ![\text{“User”}] = \text{“InvalidCardTapDueToRides”}] \\
\qquad\qquad\qquad \text{ELSE } \land\ \mathit{pc}' = [\mathit{pc} \text{ EXCEPT } ![\text{“User”}] = \text{“Done”}] \\
\qquad\qquad \land\ \text{UNCHANGED } \langle \mathit{accessAllowed}, \mathit{accessDenied}, \mathit{locked}, \mathit{timer}, \\
\qquad\qquad\qquad\qquad \mathit{cardValid}, \mathit{rides}, \mathit{counter} \rangle
\end{array}
$$

$$
\begin{array}{l}
\mathit{GrantAccess} \triangleq \land\ \mathit{pc}[\text{“User”}] = \text{“GrantAccess”} \\
\qquad\qquad \land\ \text{IF } \mathit{AccessGranted} \\
\qquad\qquad\qquad \text{THEN } \land\ \mathit{rides}' = \mathit{rides} - 1 \\
\qquad\qquad\qquad\qquad\qquad\ \land\ \mathit{accessAllowed}' = \text{TRUE} \\
\qquad\qquad\qquad\qquad\qquad\ \land\ \mathit{accessDenied}' = \text{FALSE} \\
\qquad\qquad\qquad\qquad\qquad\ \land\ \mathit{locked}' = \text{FALSE} \\
\qquad\qquad\qquad\qquad\qquad\ \land\ \mathit{timer}' = 1 \\
\qquad\qquad\qquad \text{ELSE } \land\ \text{TRUE} \\
\qquad\qquad\qquad\qquad\qquad\ \land\ \text{UNCHANGED } \langle \mathit{accessAllowed}, \mathit{accessDenied}, \mathit{locked}, \\
\qquad\qquad\qquad\qquad\qquad\qquad\qquad \mathit{timer}, \mathit{rides} \rangle \\
\qquad\qquad \land\ \mathit{pc}' = [\mathit{pc} \text{ EXCEPT } ![\text{“User”}] = \text{“WaitForReset”}] \\
\qquad\qquad \land\ \text{UNCHANGED } \langle \mathit{cardValid}, \mathit{counter} \rangle
\end{array}
$$

$$
\begin{array}{l}
\mathit{WaitForReset} \triangleq \land\ \mathit{pc}[\text{“User”}] = \text{“WaitForReset”} \\
\qquad\qquad \land\ \mathit{locked} \lor \mathit{TurstileWorkEnd} \\
\qquad\qquad \land\ \mathit{pc}' = [\mathit{pc} \text{ EXCEPT } ![\text{“User”}] = \text{“UserActions”}] \\
\qquad\qquad \land\ \text{UNCHANGED } \langle \mathit{accessAllowed}, \mathit{accessDenied}, \mathit{locked}, \mathit{timer}, \\
\qquad\qquad\qquad\qquad \mathit{cardValid}, \mathit{rides}, \mathit{counter} \rangle
\end{array}
$$

$$
\begin{array}{l}
\mathit{ValidCardTap} \triangleq \land\ \mathit{pc}[\text{“User”}] = \text{“ValidCardTap”} \\
\qquad\qquad \land\ \exists\, \mathit{rides\_num} \in 1 \mathinner{..} \mathit{MaxRidesNum} : \\
\qquad\qquad\qquad \land\ \mathit{cardValid}' = \text{TRUE} \\
\qquad\qquad\qquad \land\ \mathit{rides}' = \mathit{rides\_num} \\
\qquad\qquad\qquad \land\ \mathit{counter}' = \mathit{counter} + 1 \\
\qquad\qquad \land\ \mathit{pc}' = [\mathit{pc} \text{ EXCEPT } ![\text{“User”}] = \text{“GrantAccess”}] \\
\qquad\qquad \land\ \text{UNCHANGED } \langle \mathit{accessAllowed}, \mathit{accessDenied}, \mathit{locked}, \mathit{timer} \rangle
\end{array}
$$

$$
\begin{array}{l}
\mathit{InvalidCardTap} \triangleq \land\ \mathit{pc}[\text{“User”}] = \text{“InvalidCardTap”} \\
\qquad\qquad\qquad \land\ \exists\, \mathit{rides\_num} \in 1 \mathinner{..} \mathit{MaxRidesNum} : \\
\qquad\qquad\qquad\qquad \land\ \mathit{cardValid}' = \text{FALSE} \\
\qquad\qquad\qquad\qquad \land\ \mathit{rides}' = \mathit{rides\_num} \\
\qquad\qquad\qquad\qquad \land\ \mathit{counter}' = \mathit{counter} + 1 \\
\qquad\qquad\qquad\qquad \land\ \text{TRUE} \\
\qquad\qquad\qquad \land\ \mathit{pc}' = [\mathit{pc} \text{ EXCEPT } ![\text{“User”}] = \text{“GrantAccess”}]
\end{array}
$$

$\wedge$ UNCHANGED $\langle accessAllowed,\ accessDenied,\ locked,\ timer\rangle$

$$
\begin{aligned}
InvalidCardTapDueToRides \triangleq\ & \wedge\ pc[\text{“User”}] = \text{“InvalidCardTapDueToRides”} \\
& \wedge\ cardValid' = \text{TRUE} \\
& \wedge\ rides' = 0 \\
& \wedge\ counter' = counter + 1 \\
& \wedge\ \text{TRUE} \\
& \wedge\ pc' = [pc \text{ EXCEPT } ![\text{“User”}] = \text{“GrantAccess”}] \\
& \wedge\ \text{UNCHANGED } \langle accessAllowed,\ accessDenied, \\
& \qquad\qquad locked,\ timer\rangle
\end{aligned}
$$

$$
\begin{aligned}
user\_process \triangleq\ & UserActions \vee GrantAccess \vee WaitForReset \vee ValidCardTap \\
& \vee InvalidCardTap \vee InvalidCardTapDueToRides
\end{aligned}
$$

Allow infinite stuttering to prevent deadlock on termination.

$$
\begin{aligned}
Terminating \triangleq\ & \wedge \forall\, self \in ProcSet : pc[self] = \text{“Done”} \\
& \wedge\ \text{UNCHANGED } vars
\end{aligned}
$$

$$
\begin{aligned}
Next \triangleq\ & timer\_process \vee user\_process \\
& \vee Terminating
\end{aligned}
$$

$$Spec \triangleq Init \wedge \Box[Next]_{vars}$$

$$Termination \triangleq \Diamond(\forall\, self \in ProcSet : pc[self] = \text{“Done”})$$

END TRANSLATION

* Modification History
* Last modified Sun May 04 20:51:49 *EEST* 2025 by *Zhenia*
* Created Sat May 03 21:19:27 *EEST* 2025 by *Zhenia*