

# Introduction to UPPAAL

정세진, 손준익

# Contents

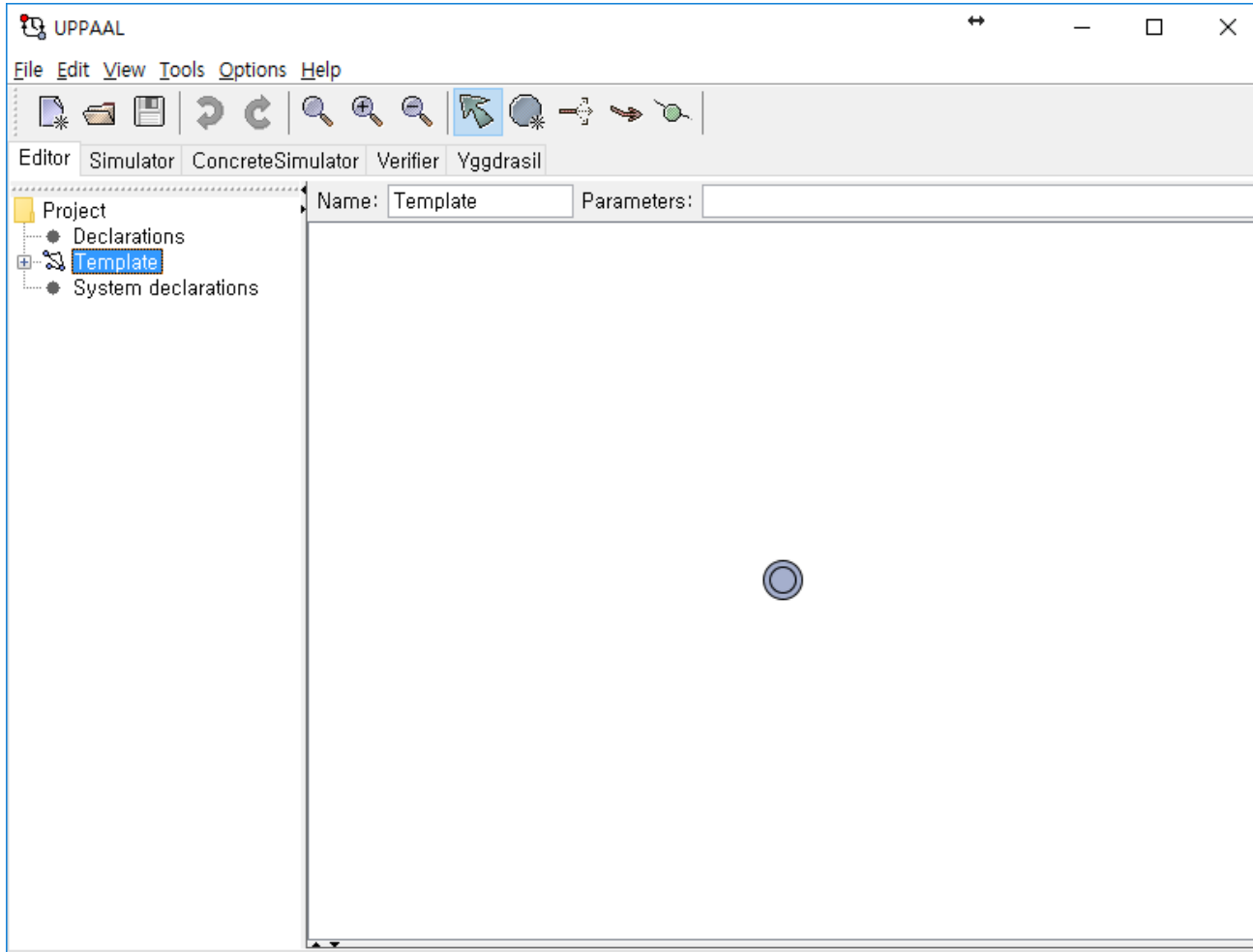
- Introduction
- Timed Automata
- Modeling
- Simulation
- Verification
- Example

# Introduction

- UPPAAL is a tool box for **validation** (via graphical simulation) and **verification** (via model-checking) of **real-time systems**
- UPPAAL은 2개의 주요 파트로 구성:
  - **Graphical User Interface (GUI)**
  - **Model-checker engine**
    - 사용자 인터페이스와 동일한 컴퓨터에서 실행 or 서버에서 독립적으로 실행 가능
- 주요 기능
  - **Modeling** : real-time system을 **timed-automata**를 이용하여 **모델링**
  - **Simulation** : 작성한 **모델의 동작**을 시뮬레이션
  - **Verification** : **TCTL** (timed computation tree logic) 표기법을 따라 입력한 **property**를 모델이 만족하는지 **확인**

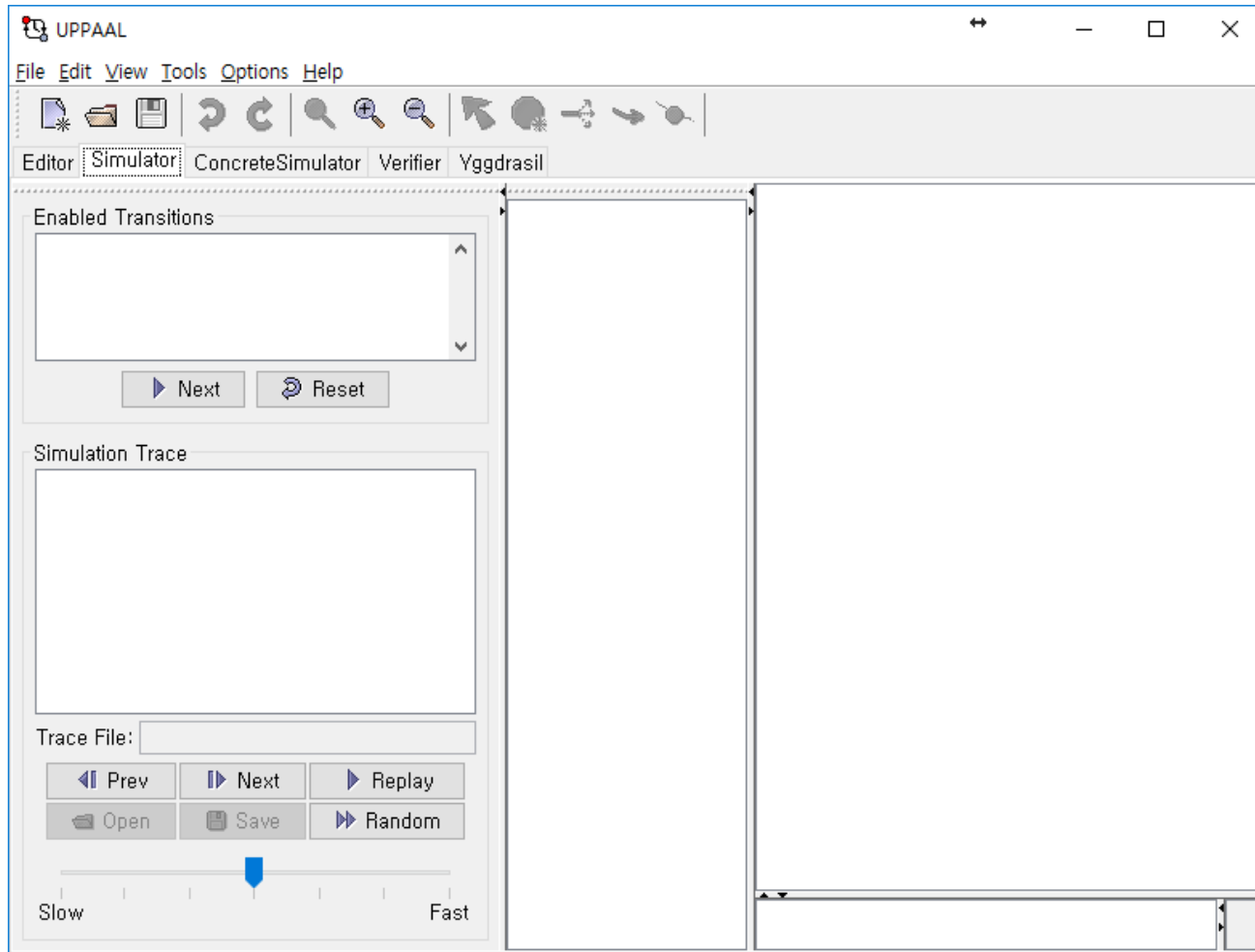
# Introduction

- Graphical User Interface - Editor



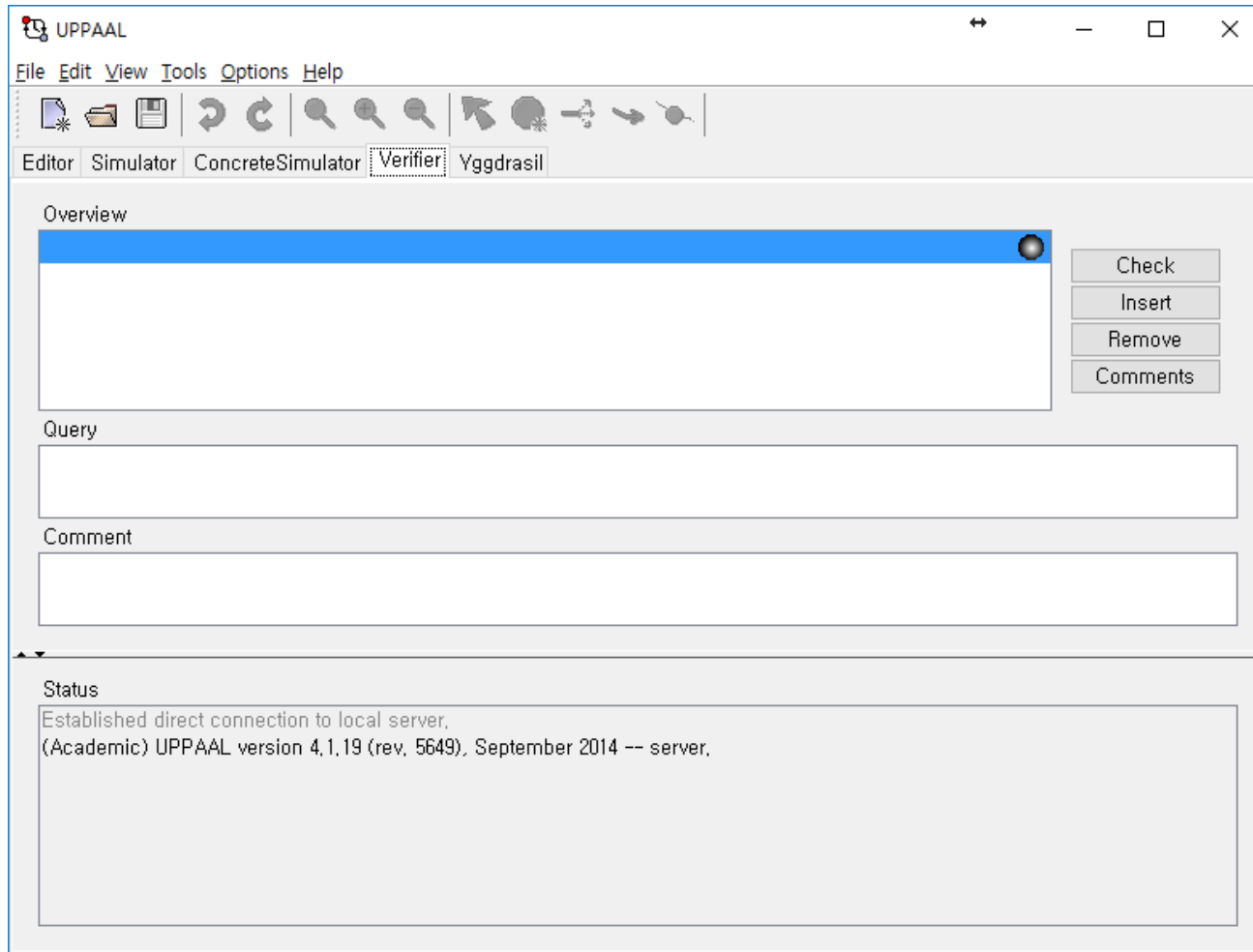
# Introduction

- Graphical User Interface - Simulator



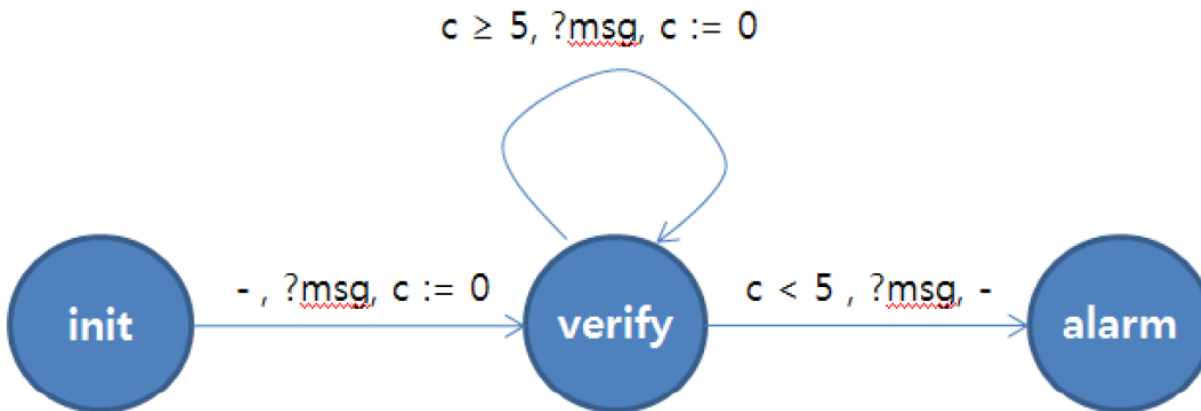
# Introduction

- Graphical User Interface - Verifier



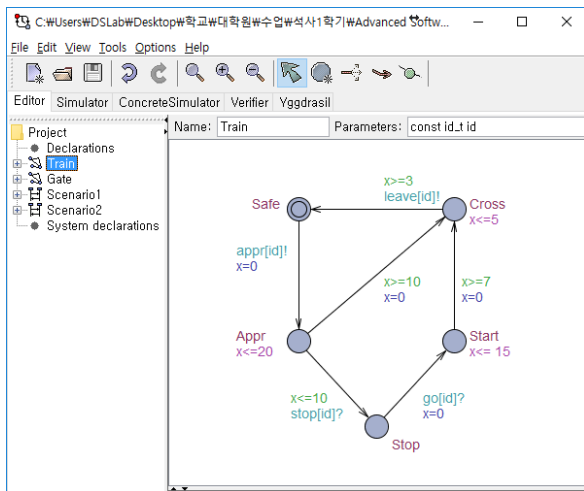
# Timed Automata

- Timed Automata = Finite State Machine + Clock
  - Clock은 연속적인 시간, 즉 실수 값을 가짐
  - Clock은 시스템의 global clock과 각 모듈의 local clock이 존재
  - 모든 Clock은 동일한 속도로 증가
  - Clock은 Transition의 guard에서 조건으로 사용이 가능하며, Transition에서 reset 될 수 있음
- Invariant : 항상 만족해야 하는 clock 값에 대한 상태의 조건

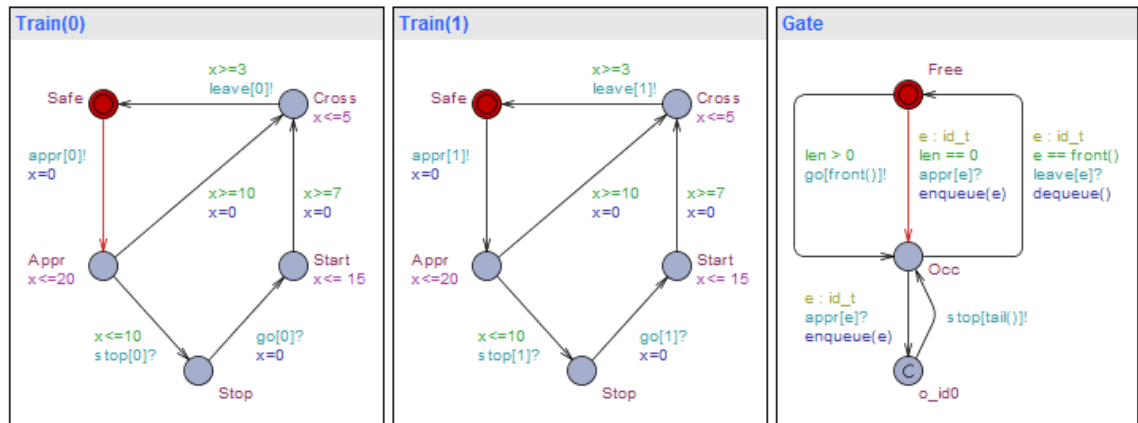


# Modeling

- In UPPAAL, 시스템은 **timed-automata**를 이용하여 모델링됨
  - 각 automaton의 clock은 동일한 속도로 증가
- **Structure of an UPPAL Model**
  - UPPAAL 모델은 동시에 실행되는 프로세스들의 집합으로 구성
  - 각각의 프로세스는 **timed-automaton**으로 디자인
    - Automaton은 **location**의 집합과 **edge**를 가지는 graph로 표현
  - 동일한 automaton의 인스턴스화를 사용하기 위해 **Template**을 사용



Template



Set of Process



# Modeling

- **Location**은 이름과 **Invariant**를 가질 수 있음
- Location은 4가지 타입이 존재
  - Normal
  - Initial
  - Urgent
  - Committed

The screenshot shows a dialog box titled "Edit Location" with a close button (X) in the top right corner. The dialog has three tabs: "Location", "Comments", and "Test Code". The "Location" tab is selected. Inside the dialog, there are the following fields and controls:

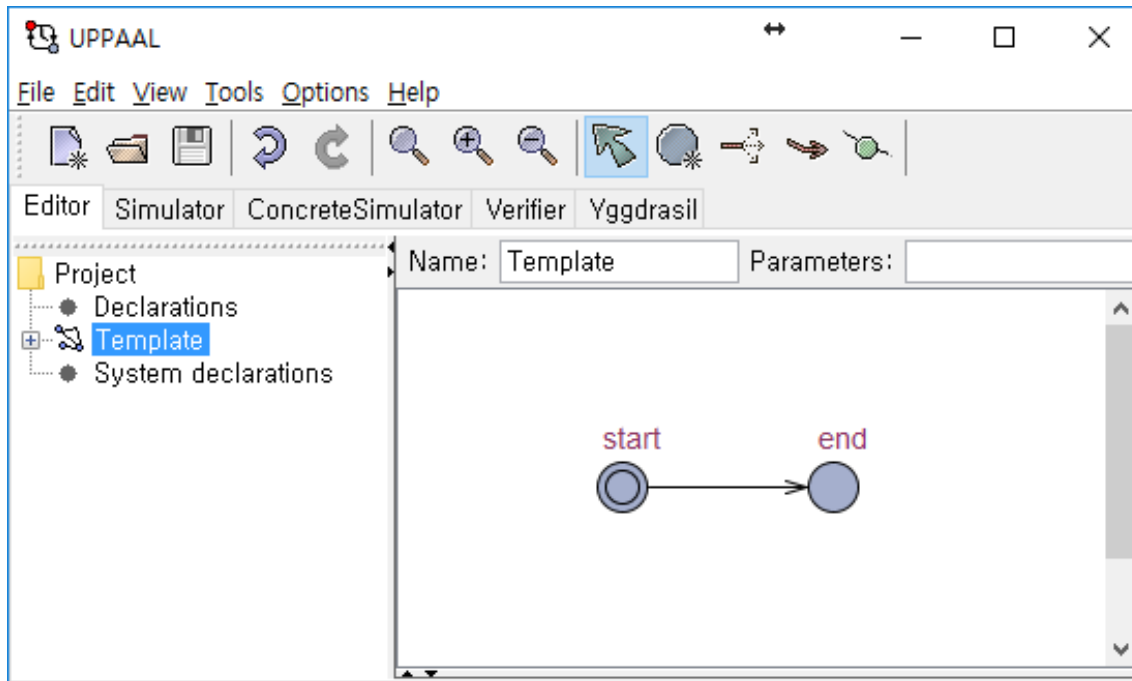
- Name:** A text input field.
- Invariant:** A larger text area for entering the invariant.
- Rate of Exponential:** A text input field.
- Initial:** A checked checkbox.
- Urgent:** An unchecked checkbox.
- Committed:** An unchecked checkbox.

At the bottom of the dialog, there are two buttons: "OK" and "Cancel".

# Modeling

- Initial location

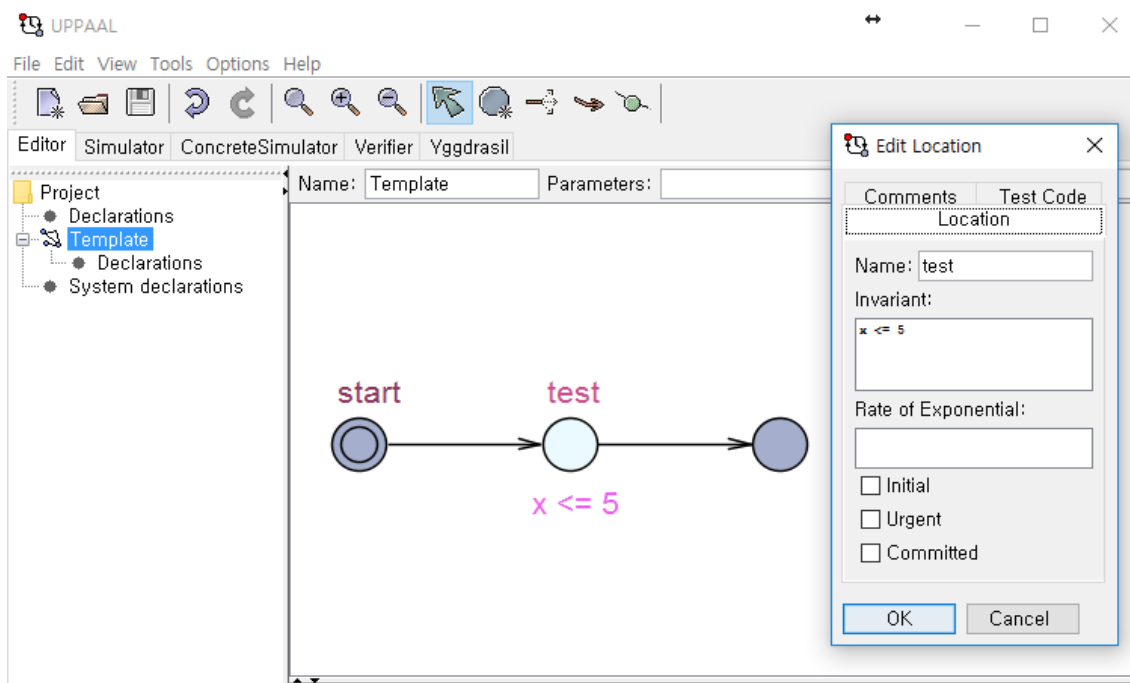
- 템플릿은 특정 위치에서 시작해야 해야함
- 각 템플릿은 initial로 표시된 위치가 하나 존재 해야함
- Initial location은 이중 원으로 표현



# Modeling

## • Invariant

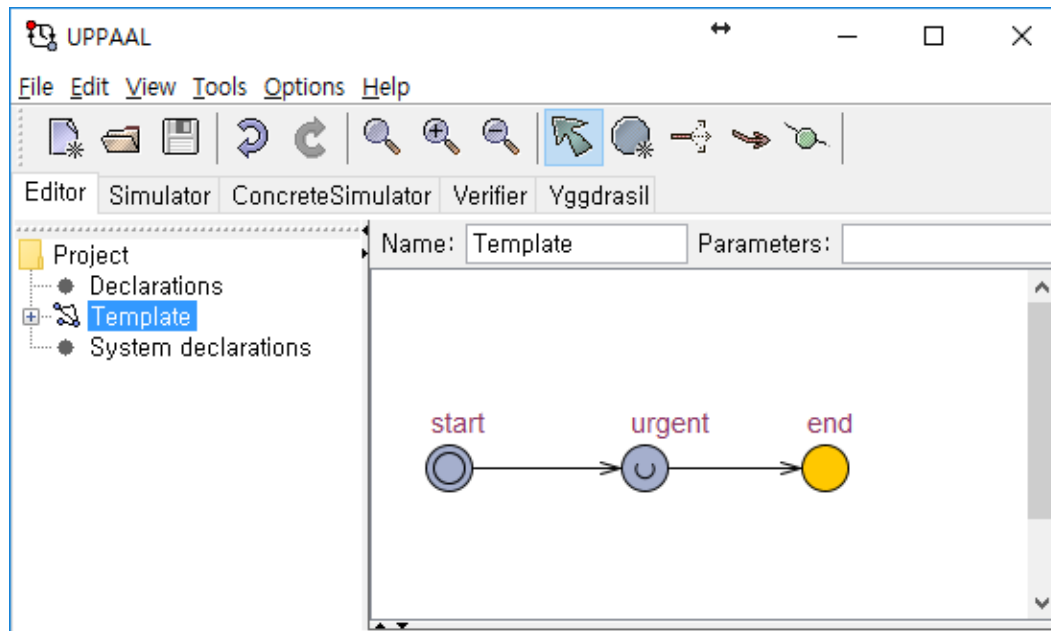
- Invariant는 automaton이 해당 location에 있는 동안 무조건 만족되어야 하는 조건
- Initial, Normal location은 invariant를 가질 수 있음
- 각 location의 Invariant는 clock에 대한 제약 조건을 표시



# Modeling

- **Urgent location**

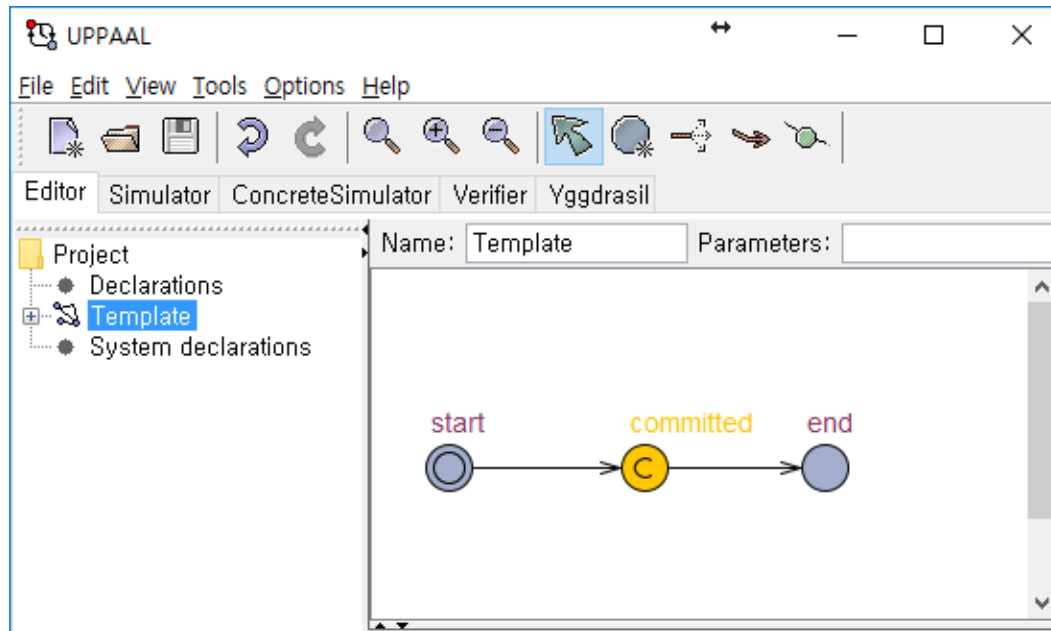
- Urgent location에서는 시간이 흐르지 않음
- 시간이 지나기 전에 urgent location에서 전이가 일어나야 함
- 동시에 실행중인 다른 프로세스에서는 Urgent location에서 전이가 일어나기 전에 전이가 발생할 수 있음
- Urgent location은 "U"로 표현



# Modeling

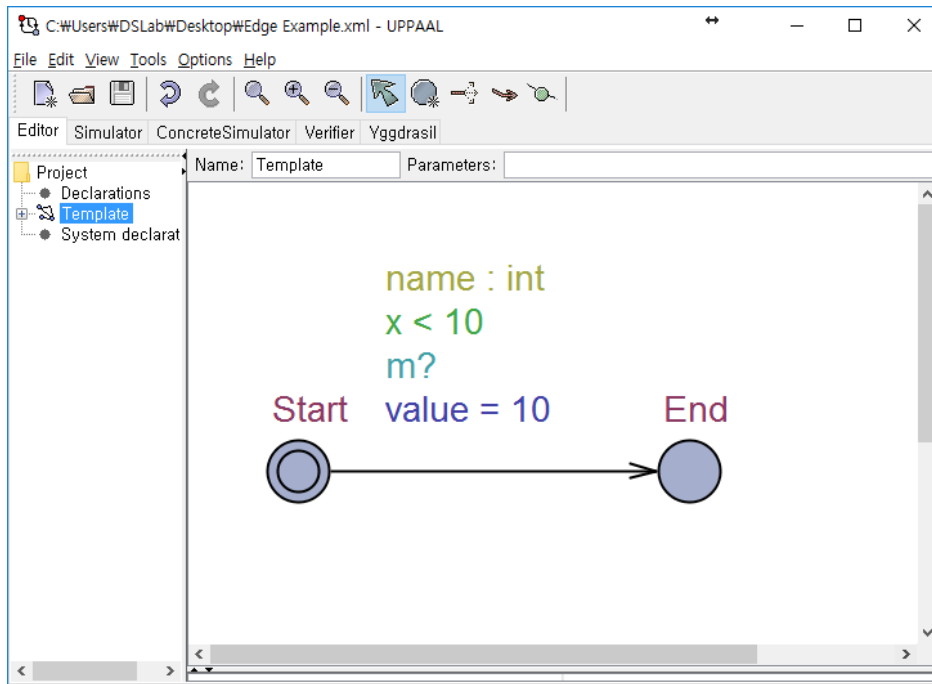
- **Committed location**

- Committed location에서는 시간이 흐르지 않음
- 동시에 실행중인 다른 프로세스에서는 Committed location에서 전이가 일어나기 전까지는 전이가 발생할 수 없음
  - Committed location은 atomic sequence를 만드는데 유용함
- Committed location은 "C"로 표현



# Modeling

- Edge는 **Guard, Update, Select and Synchronization** 로 labeling 됨
  - 각각의 label은 필수가 아니라 선택적



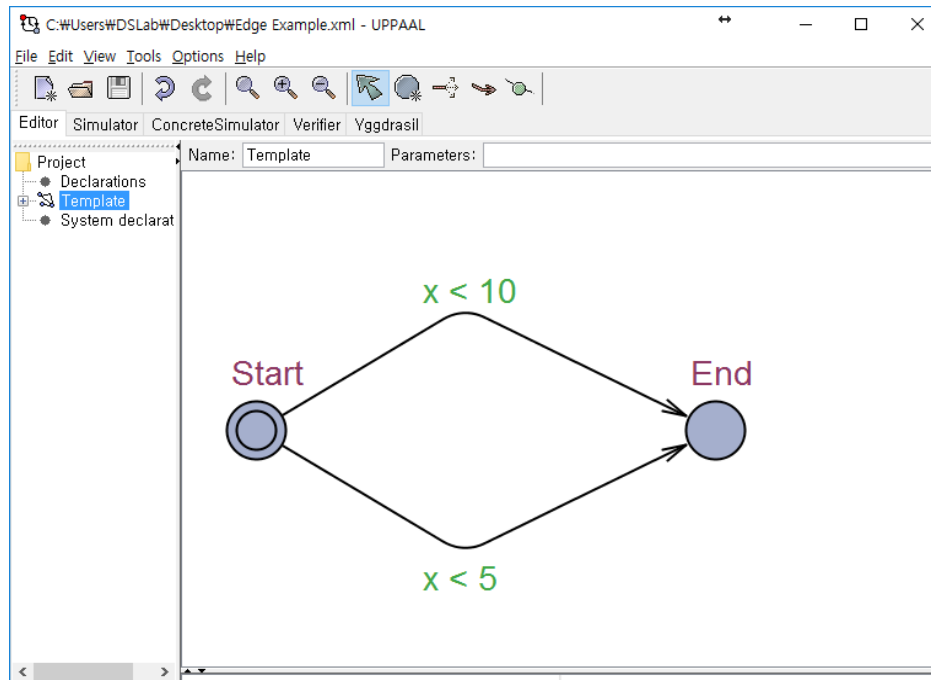
The screenshot shows the 'Edit Edge' dialog box in UPPAAL. The dialog has three tabs: 'Edge', 'Comments', and 'Test Code'. The 'Edge' tab is active, showing fields for 'Select', 'Guard', 'Sync', and 'Update'. The 'Select' field contains `name : int`, 'Guard' contains `x < 10`, 'Sync' contains `m?`, and 'Update' contains `value = 10`. There are 'OK' and 'Cancel' buttons at the bottom.

# Modeling

- **Guard**는 모델의 clock과 variable을 사용한 표현식으로 언제 전이가 가능한지 나타내기 위해 사용

ex)  $x > 0$ ,  $x == 10$

- 특정 시간에 여러 개의 edge가 guard를 만족하는 경우 그 중 하나의 edge만 전이가 발생함



# Modeling

- **Update**는 전이가 일어났을 때 시스템의 상태를 변화시키기 위해 사용
  - 표현식은 Clock, integer variables, constant만을 사용  
clock에는 정수 값만 할당  
ex) clock = 0, set = true,
- **Select**는 edge에서만 사용되는 임의의 값을 갖는 변수를 사용하기 위해 사용
  - name : type 으로 표현
    - name : 변수의 이름
    - type : 정의된 type (built-in or custom)



# Modeling

- **Synchronization** 은 둘 이상의 프로세스의 동작을 조정하는 데 사용되는 기본 메카니즘으로 Message Passing framework를 따름
  - 둘 또는 그 이상의 프로세스가 동시에 전이를 발생
    - **m!** : Emitting a message
    - **m?** : Reception of the message

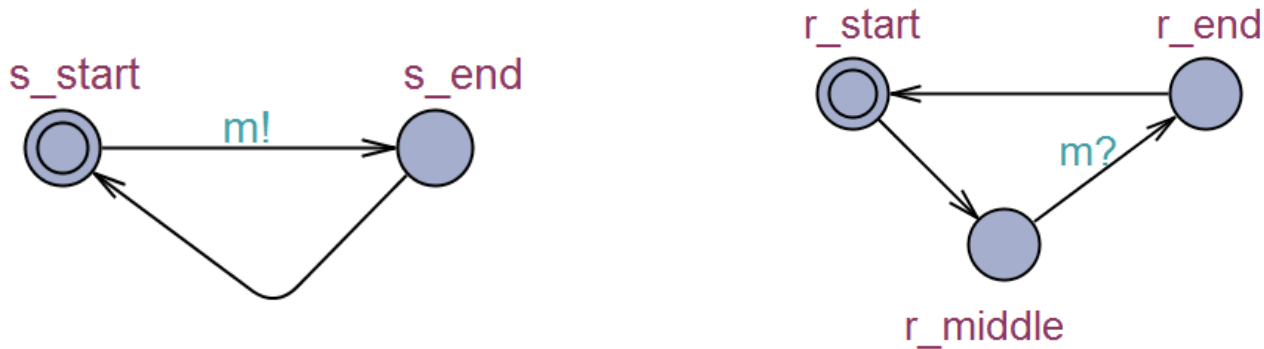


- UPPAAL에는 3가지 다른 synchronization이 존재
  - Regular channel
  - Urgent channel
  - Broadcast channel

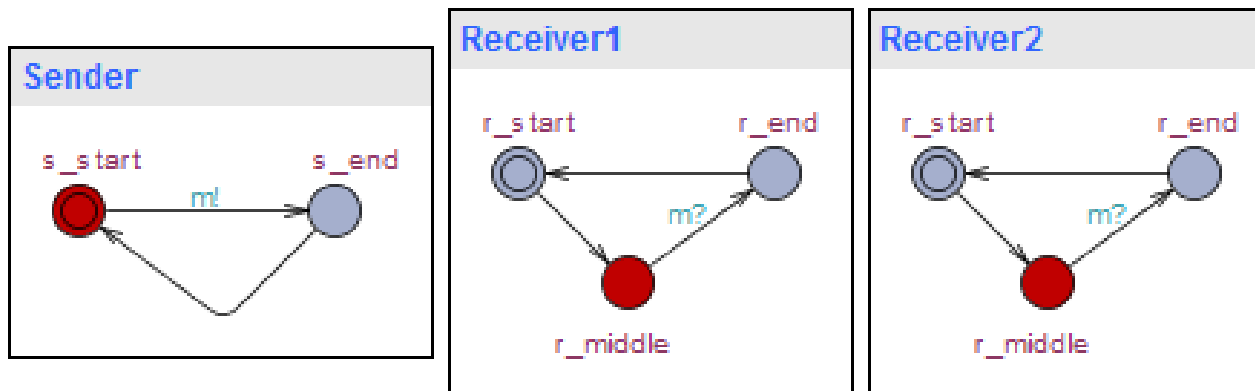
# Modeling

- **Regular channel**

- Declaration에서 다음과 같이 선언, e.g. chan c;
- 메시지 emission과 reception이 동시에 만족하는 경우 동시에 전이가 발생



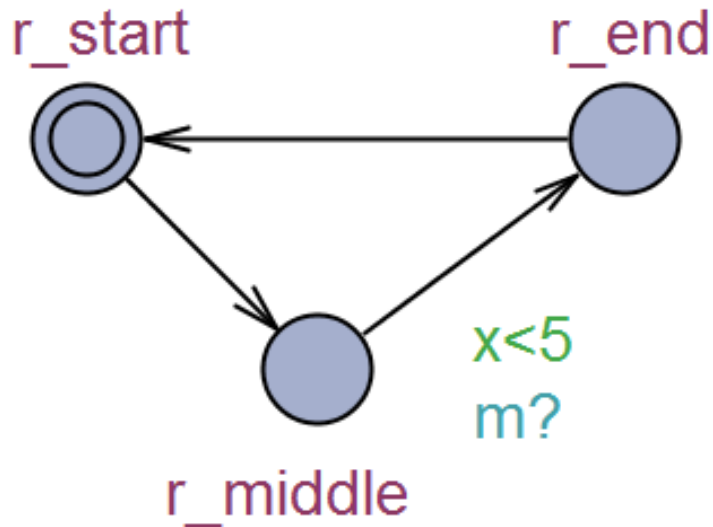
- 동시에 여러 프로세스가 가능할 경우 non-deterministic 하게 하나의 쌍만 전이가 발생



# Modeling

- **Urgent channel**

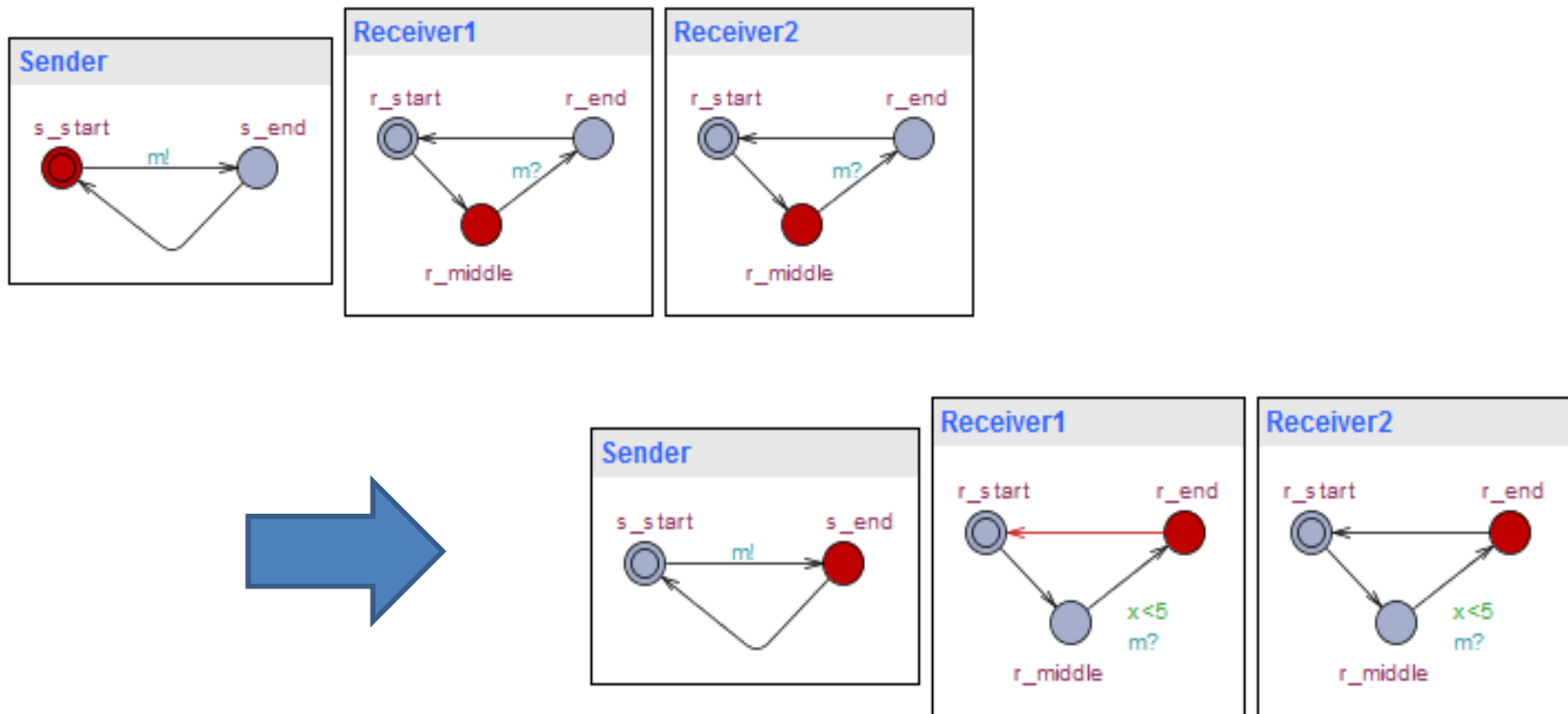
- Declaration에서 다음과 같이 선언, e.g. urgent chan c;
- Regular channel과 거의 유사
- 두 프로세스가 이미 동기화 될 수 있는 상황에서 지연을 방지하기 위해 사용
- urgent channel이 있는 edge에서는 guard에 clock이 허용되지 않음
  - 아래와 같은 경우, m에 대해서 regular channel은 가능하지만 urgent channel은 불가능



# Modeling

- **Broadcast channel**

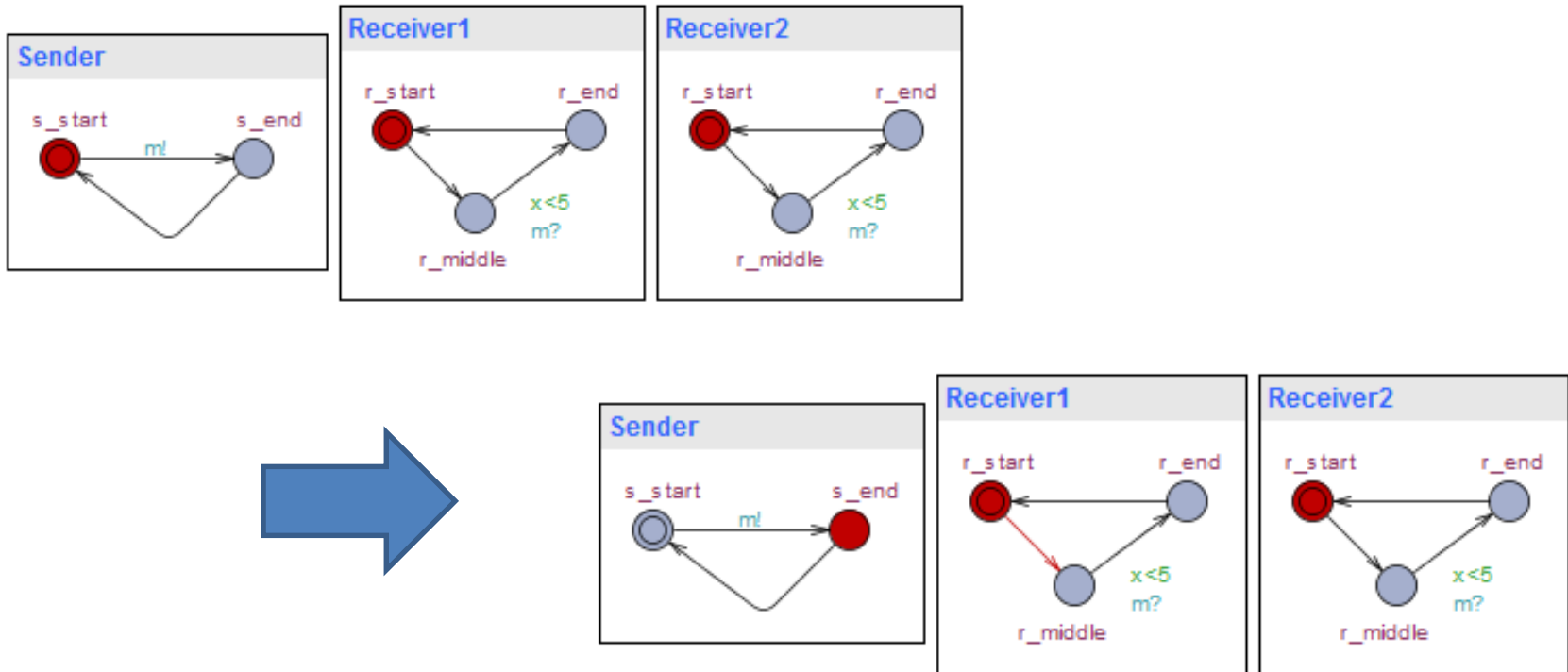
- Declaration에서 다음과 같이 선언, e.g. broadcast chan c;
- 동시에 여러 프로세스가 가능 할 경우, 가능한 모든 프로세스에서 전이가 발생



# Modeling

- Broadcast channel

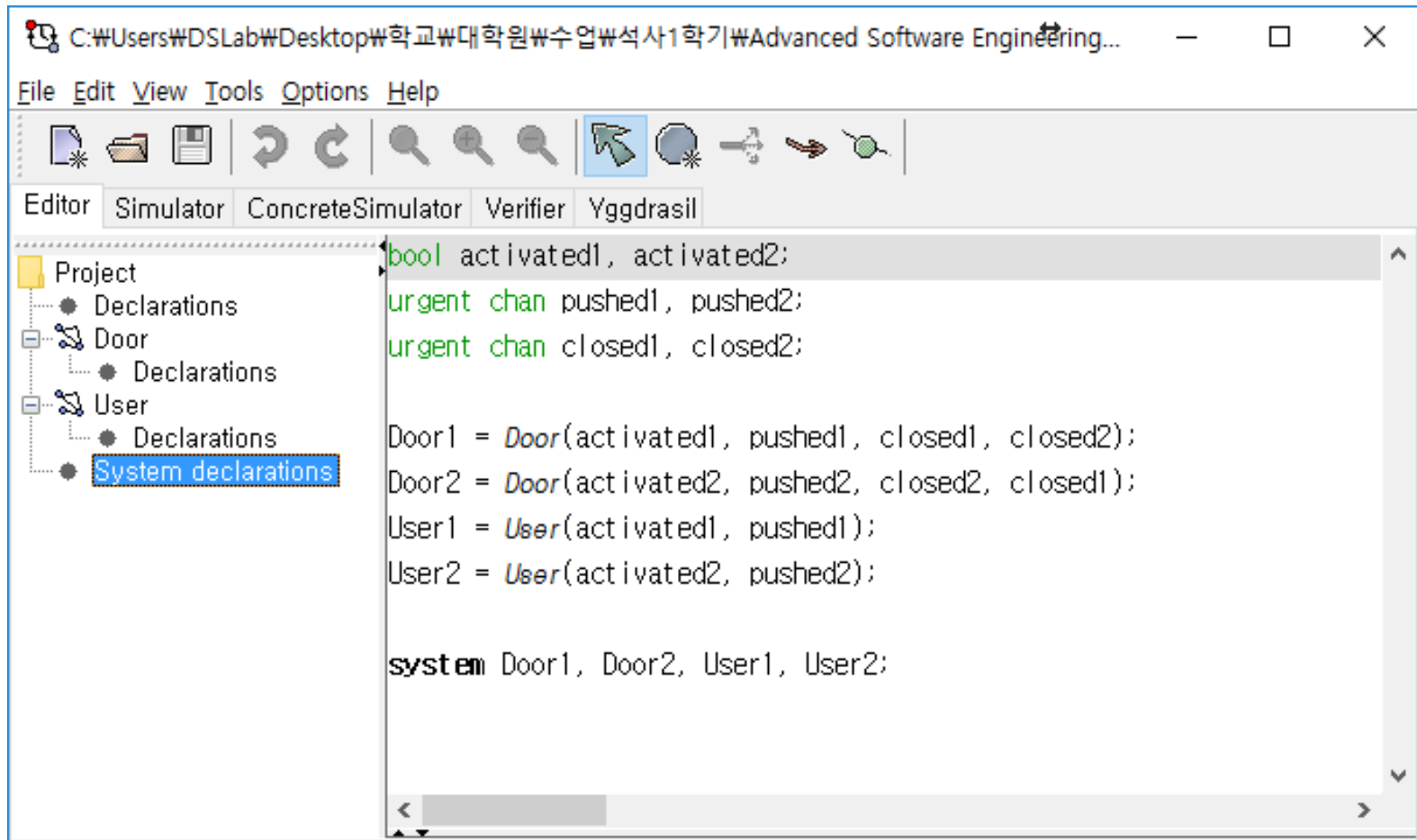
- 메시지를 emission하는 프로세스만 있는 경우에도 전이가 발생



- urgent channel과 마찬가지로 edge의 guard에 clock이 허용되지 않음

# Modeling

- System declarations



The screenshot shows a software development environment window titled "C:\Users\WDSLab\Desktop\학교부대학원부수업부석사1학기\Advanced Software Engineering...". The window has a menu bar with "File", "Edit", "View", "Tools", "Options", and "Help". Below the menu bar is a toolbar with various icons. The main area is divided into two panes. The left pane is a project explorer showing a tree structure: "Project" (expanded) contains "Declarations", "Door" (expanded), "User" (expanded), and "System declarations" (selected and highlighted). The right pane is a code editor showing the following code:

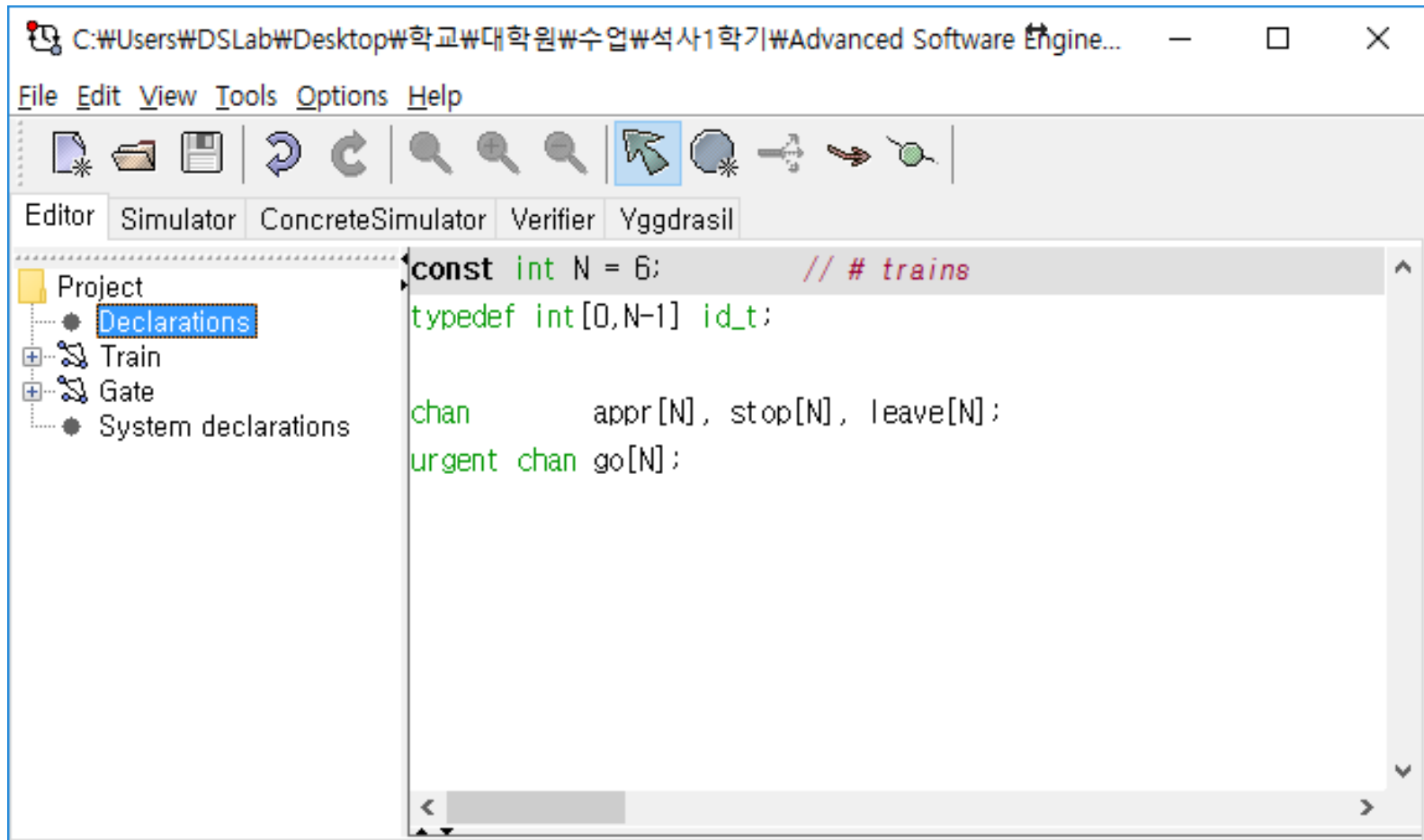
```
bool activated1, activated2;
urgent chan pushed1, pushed2;
urgent chan closed1, closed2;

Door1 = Door(activated1, pushed1, closed1, closed2);
Door2 = Door(activated2, pushed2, closed2, closed1);
User1 = User(activated1, pushed1);
User2 = User(activated2, pushed2);

system Door1, Door2, User1, User2;
```

# Modeling

- Global Declarations



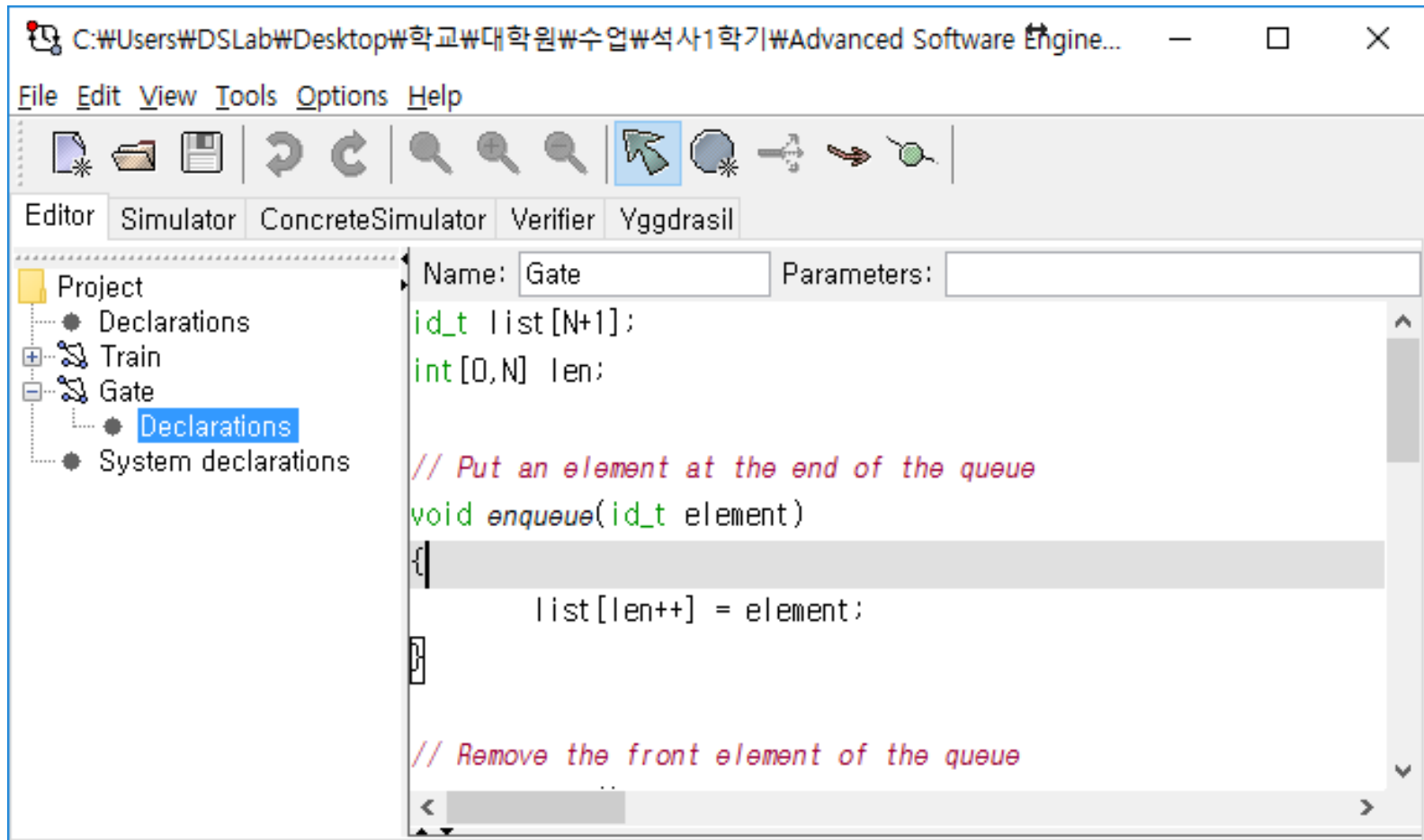
The screenshot shows the Advanced Software Engine IDE interface. The title bar indicates the file path: C:\Users\DSL\Lab\Desktop\학교\대학원\수업\석사1학기\Advanced Software Engine... The menu bar includes File, Edit, View, Tools, Options, and Help. The toolbar contains icons for file operations, simulation, and search. The main window is divided into a left sidebar and a central editor. The sidebar shows a project tree with 'Declarations' selected. The editor displays the following code:

```
const int N = 6;           // # trains
typedef int [0,N-1] id_t;

chan      appr [N], stop [N], leave [N];
urgent chan go [N];
```

# Modeling

- Template Declarations



The screenshot shows the Yggdrasil software interface. The window title is "C:\Users\DSLab\Desktop\학교\대학원\수업\석사1학기\Advanced Software Engine...". The menu bar includes "File", "Edit", "View", "Tools", "Options", and "Help". The toolbar contains various icons for file operations and simulation. The "Editor" tab is active, showing a project tree on the left with "Project", "Declarations", "Train", "Gate", "Declarations", and "System declarations". The main editor area displays the following code:

```
Name: Gate Parameters:
id_t list [N+1];
int [0,N] len;

// Put an element at the end of the queue
void enqueue(id_t element)
{
    list[len++] = element;
}

// Remove the front element of the queue
```



# Simulation

- 정의한 System의 행동을 단계 별로 확인
  - 단계별 각 프로세스의 상태와 변수 값 확인 가능

The screenshot displays the UPPAAL simulation environment. The main window is titled "C:\Users\DSLab\Desktop\학교#대학원#수업#부석사1학기#Advanced Software Engineering#UPPAL#uppal-4.1.19#demo#lsc\_train-gate\_parameters.xml - UPPAAL".

**Left Panel:** Contains the "Enabled Transitions" list (currently showing "Train(0)"), "Simulation Trace" (listing transitions like "Train(1)", "appr[0]: Train(0) → Gate[0]", etc.), and "Trace File" controls.

**Center Panel:** Shows the state of three processes:
 

- Train(0):** States include Safe, Appr, Stop, and Cross. Transitions are labeled with conditions like  $x \geq 3$ ,  $x \leq 10$ , and  $x \leq 15$ .
- Train(1):** States include Safe, Appr, Stop, and Cross. Transitions are labeled with conditions like  $x \geq 3$ ,  $x \leq 10$ , and  $x \leq 15$ .
- Gate:** States include Free, Occ, and o\_id0. Transitions are labeled with conditions like  $len > 0$  and  $len == 0$ .

**Bottom Panel:** A state transition diagram showing the sequence of events between the processes, with labels like "leave[1]", "go[front()]", "appr[1]", and "stop[tail()]" indicating the flow of control and data.

# Simulation

- **Simulator**는 3가지 방법으로 사용 가능

1. 사용자가 그 다음 단계로 취할 전이를 선택하면서 수동으로 시스템을 시뮬레이션
2. 시스템이 랜덤하게 자체적으로 시뮬레이션
3. verification을 이용하여 특정 속성에 대한 reachable state를 추적하는 시뮬레이션

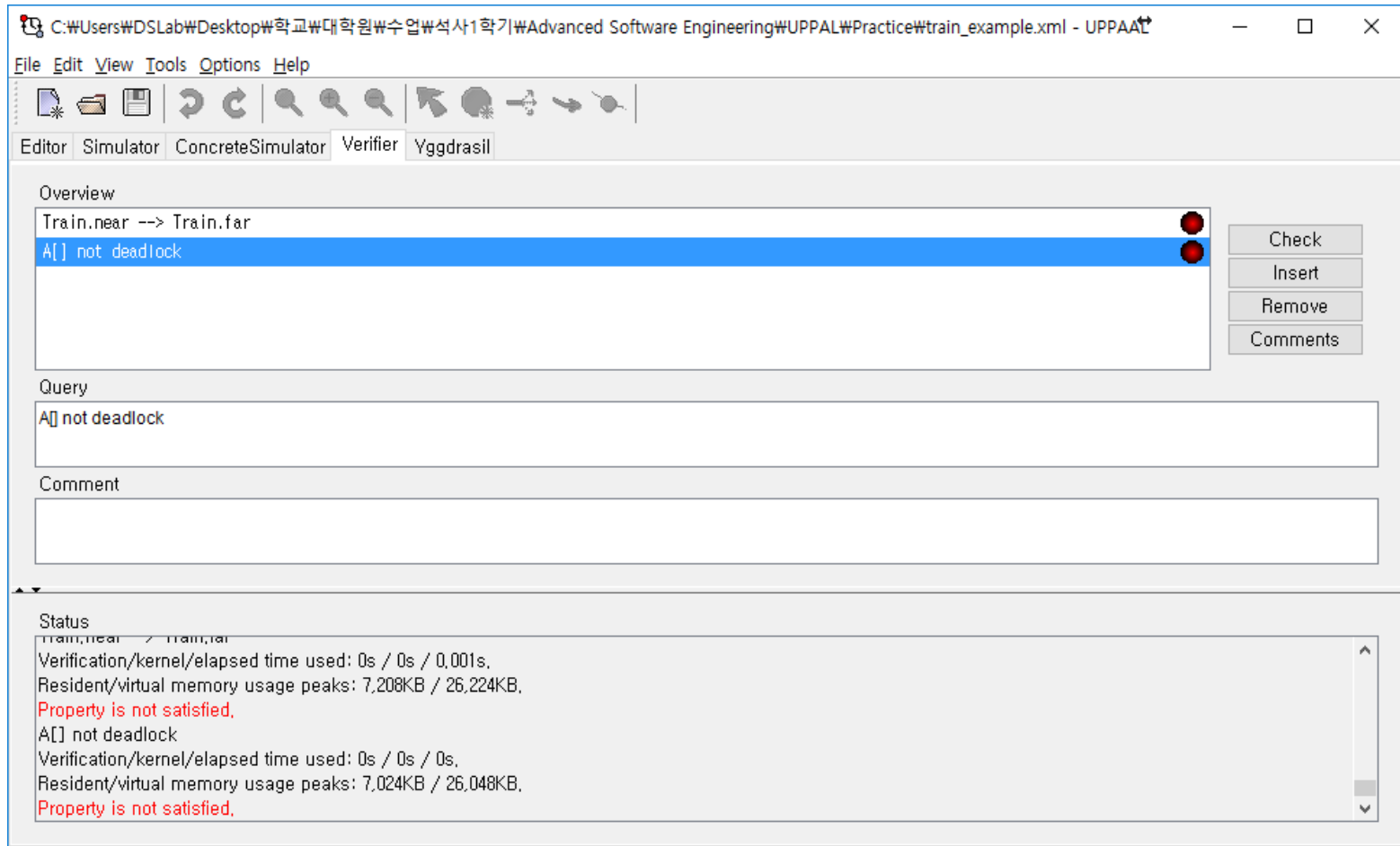
1

The screenshot shows the UPPAAL simulator interface. On the left, the 'Enabled Transitions' list is highlighted with a red box, showing 'Train(0) appr[1]: Train(1) -> Gate[1]'. Below it, the 'Simulation Trace' shows '(Safe, Safe, Free)' and 'appr[0]: Train(0) -> Gate[0]'. At the bottom left, the 'Random' button is highlighted with a red box. The main area displays three state transition diagrams for Train(0), Train(1), and Gate, and a corresponding physical representation of the train and gate states.

2

# Verification

- 특정 property를 시스템이 만족하는지 확인 – Model Checking

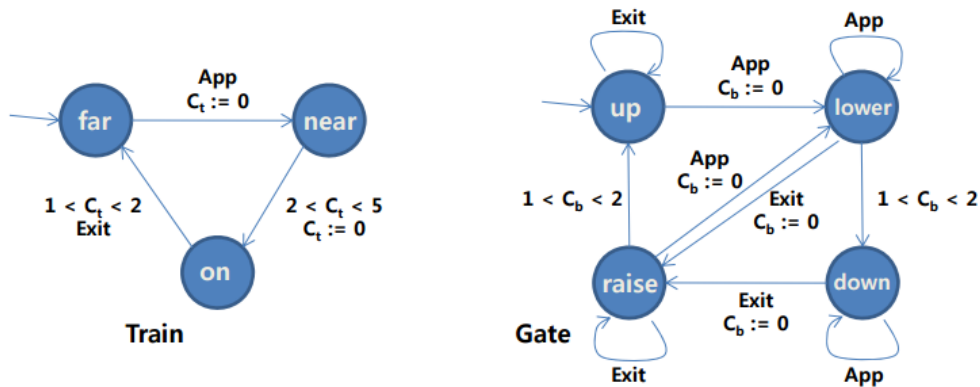
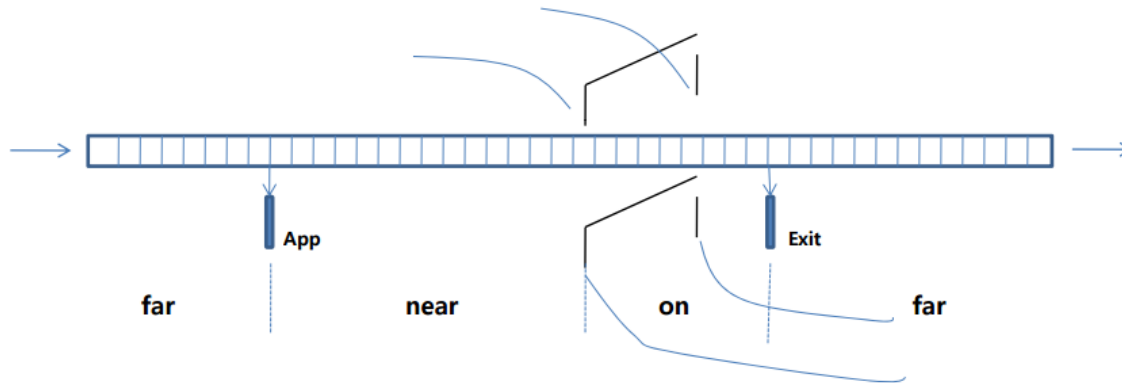


# Verification

- Syntax and Semantic of Properties
  - $E\langle\rangle p$  : there exists a path where  $p$  eventually holds. (= EF  $p$ )
  - $E[ ] p$  : there exists a path where  $p$  always holds. (= EG  $p$ )
  - $A\langle\rangle p$  : for all paths  $p$  will eventually holds. (= AF  $p$ )
  - $A[ ] p$  : for all paths  $p$  always holds. (= AG  $p$ )
  - $p \rightarrow q$  : whenever  $p$  holds  $q$  will eventually hold.
  - $E\langle\rangle \text{deadlock}$  : Exists deadlock
  - $A[ ] \text{not deadlock}$  : There is no deadlock
- 템플릿의 인스턴스에 `dot(.)`을 사용하면 location 및 템플릿 변수에 접근 가능  
Ex) `Train.near`  $\rightarrow$  `Train.far`

# Example

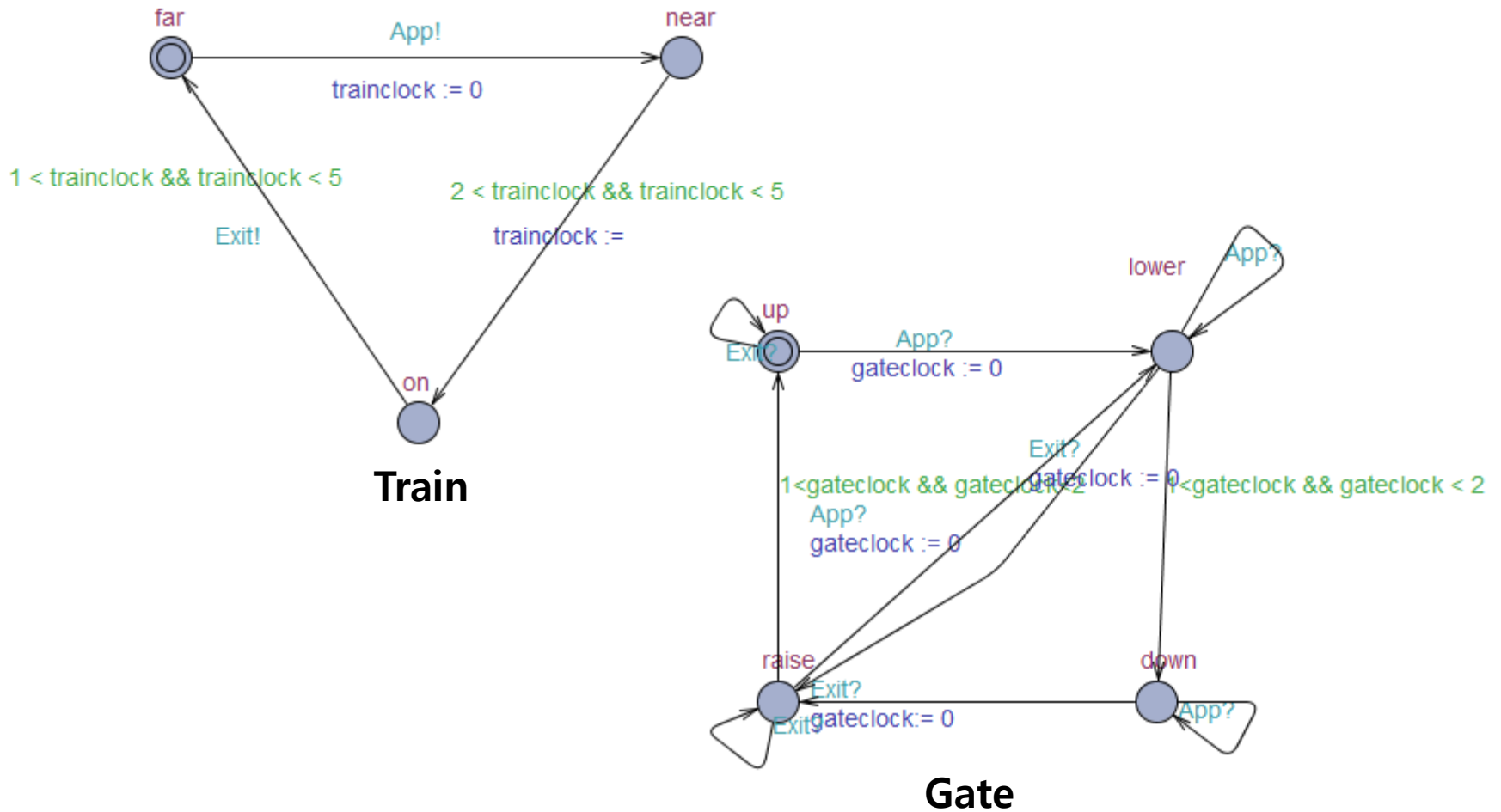
- Railroad examples in lecture note



- Example : modeling a railroad crossing

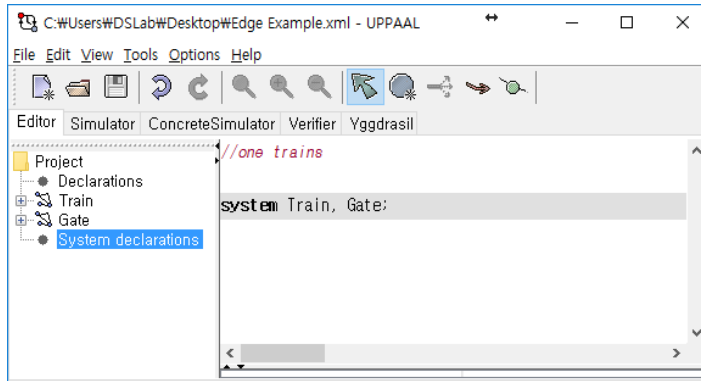
# Example

- Railroad examples in lecture note

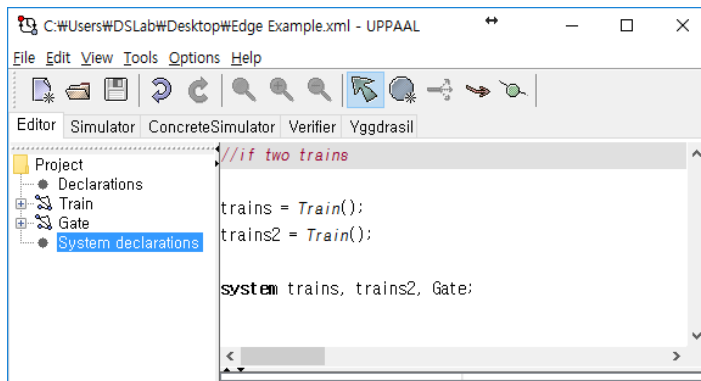


# Example

- System Declaration
  - If one trains



- If two trains



# Example

- Simulation
  - Lower -> Raise로 바로 transition 발생
  - Gateclock이 guarding condition을 넘어감

The screenshot displays the UPPAAL simulator interface. The main window shows two state transition diagrams: 'Train' and 'Gate'. The 'Train' diagram has states 'far', 'near', and 'on'. Transitions include 'App?' from 'far' to 'near' (guarding condition:  $1 < \text{trainlock} \ \&\& \ \text{trainlock} < 5$ ) and 'Exit' from 'near' to 'on' (guarding condition:  $2 < \text{trainlock} \ \&\& \ \text{trainlock} < 5$ ). The 'Gate' diagram has states 'up', 'down', and 'lower'. Transitions include 'App?' from 'up' to 'lower' (guarding condition:  $! \text{gateclock} \ \&\& \ \text{gateclock} = 0$ ) and 'App?' from 'down' to 'lower' (guarding condition:  $! \text{gateclock} \ \&\& \ \text{gateclock} < 2$ ). The 'Simulation Trace' window shows a sequence of transitions: (near, lower), Gate, (near, down), Train, (on, down), Exit: Train -> Gate, (far, raise), App: Train -> Gate, (near, lower), Train, (on, lower), Exit: Train -> Gate, (far, raise). A red box highlights the 'Exit: Train -> Gate' transition. The 'Global variables' window shows `gateclock = 4.147816`, also highlighted with a red box.



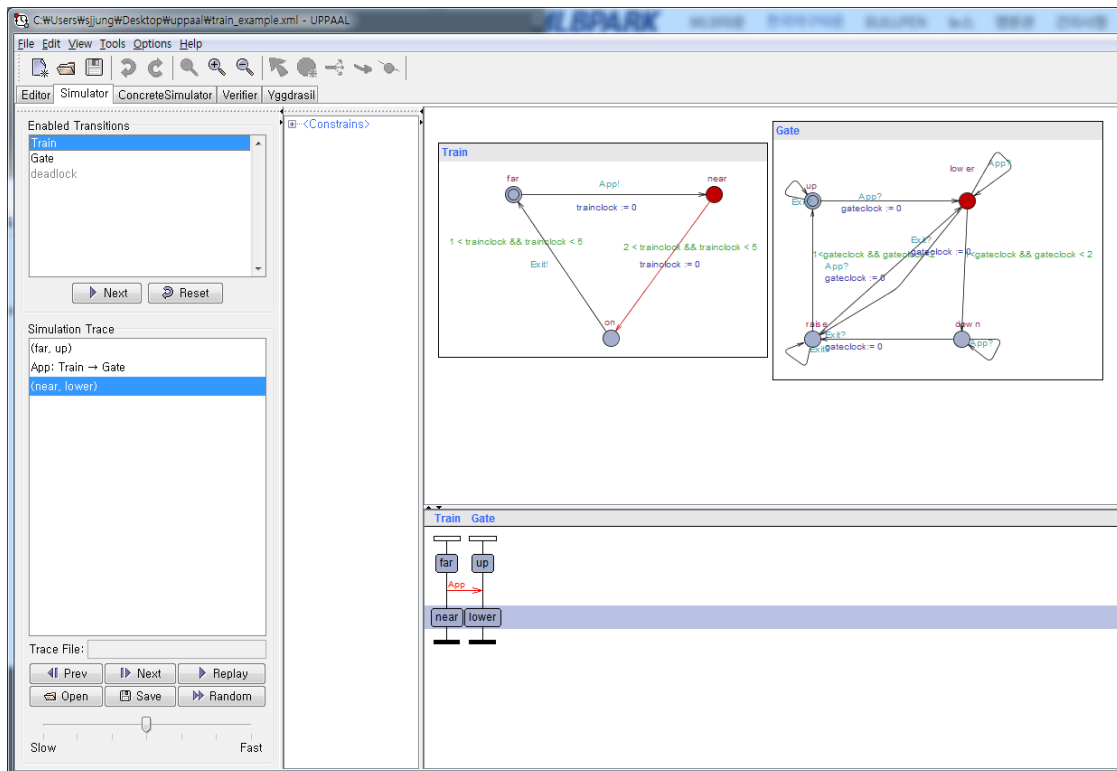
# Example

- Verification

- Property :  $A[] \text{ !deadlock}$

- Unsatisfied : Lower location 에서 invariants가 없어서 deadlock이 될 수 있다고 추측

- Counter example



# Example

- Verification
  - Property : Train.near --> Train.far
  - Unsatisfied : 위와 같은 이유로 추측
  - Counter example

