Universidade do Minho

Mestrado Integrado em Engenharia Informática - 4º ano

Métodos Formais em Engenharia de Software

2016/2017

Especificação e Modelação
Modelling an Aircraft Landing System

David Moreira
Fernanda Alves

February 3, 2017

Contents

1 Introduction

1.1 Contextualization

In order to specification, modelling and reasoning for the production of high reliable software we develop this report about modelling and Aircraft Landing System in Alloy with the purpose of applying this knowledge learned along the semester.

The main idea is to study this same modelling in Event-B and transposes this old model to Alloy.

1.2 Case Study

Our case study, mentioned before, is about Modelling and Aircraft Landing System in Alloy. Obviously, the Landing System is a crucial part of all Aircraft System and can lead to loss of lives, so we have the challenge of verifying and validate an avionic system in Alloy.

Therefore, the Landing System is composed of three main components: mechanical system, digital system, and pilot interface. The mechanical system is composed of three landing sets, the digital system is the responsible for controlling mechanical parts and the pilot interface is the bridge between the pilot and rest of the system that has a handle and a set of indicators.

For this case study, we work about each component modelling and specify each characteristic of all three. In order to understand our way of thinking and how the component modelled works, along with the report we will explain and discriminate these implementations.

It should be noted that we follow the order of the article which has been ceded to us.

2 Formal Development of the Landing System

2.1 Pilot Interface

The pilot interface has an Up/Down handle and a set of indicators. The handle is used by pilot for extending or retracting the landing gear sequence, and a set of indicators is the different type of lights for giving the actual position of gears and doors, and the system state. Henceforth, we will explain the modelling of all these features available in the Pilot Interface.

2.1.1 Up and Down Handle

For the practical purpose of Landing System, the feature of being halted up and retracted or halted down and extended as we show in the figure, the pilot have the Up/Down handle. For this events exists a button which has PressDOWN and PressUP, respectively, that are enabled when the pilot want to change the state of the landing system to another.

Therefore, we will show our modelling for this features.

```
sig PilotInterface {
    button : Bstate -> State,
    control : one DigitalSystem,
    trigger : one AnalogicSystem,
    flightSensor : OFstate -> State
}

abstract sig Bstate {}

one sig Up, Down extends Bstate {}

abstract sig OFstate {}

one sig On, Off extends OFstate {}
```

Basically, the Pilot Interface will have this button to control the state of landing system. In this situation, our modelling setup two possible states to the button: UP and DOWN, marked ”one”, in other words, there will always be only one instance of it.

Also, we ignore the time of moving down and moving up as well the possibility of turning state back during the moving.

2.2 The Mechanical System

The Mechanical System is composed of three landing sets, where each set contains landing gear box and a door with latching boxes. The landing gears and doors motions are performed with the help of cylinders. These electro-valves are activated by a digital system.

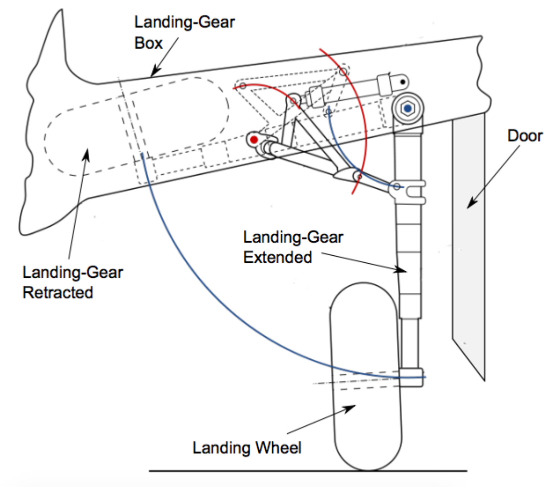

Figure 1: Landing Gear System

2.2.1 Landing Set

To change the states previously described, exist some proprieties and behaviours that must be safely granted. These movements affect the system and we have to distinguish that doors are alternatively open and unlocked or closed and locked.

```
abstract sig Lstate {}

one sig Lock, Unlock extends Lstate {}

abstract sig Dstate {}

one sig Open, Close extends Dstate {}

abstract sig Hstate {}

one sig Extend, Retract extends Hstate {}
```

As it can be seen, the same reasoning can be used in the context of the doors and hydraulic system. For each system, there is the possibility to have only one state: lock or unlock for door lock state, open or closed for door state and retracted or extended for the hydraulic state.

All these subsystems are part of each of three Landing Set that forms all Landing System. In order to set this, we create a sig named LandingSet as shown in next code.

```
sig LandingSet {
    landingset : Lstate -> State,
    valvesDoor : one Door,
    valvesHydraulic : one Hydraulic
}

abstract sig Door {
    door : set Dstate -> State
}

abstract sig Hydraulic {
    hydraulic : set Hstate -> State
}
```

2.3 Digital System

The digital system is only the responsible for controlling mechanical system and for detecting anomalies so, consequently, this system will control the previous section revealed. Furthermore, we define the detecting of anomalies and respective Analogic System that activate the emergency mode when any anomaly is detected in the system.

```
sig DigitalSystem {
 anomaly : DSstate -> State,
 emergency : one AnalogicSystem,
 handle : one LandingSet
}

abstract sig DSstate {}

one sig Detect, Undetect extends DSstate {}

abstract sig AnalogicSystem {
 manual : ASstate -> State
}

abstract sig ASstate {}

one sig Active, Inactive extends ASstate {}
```

2.4 Verification and Validation of the System

In order to ensure the model functionality and security, there are standards that must be covered. For this reason, all of the modellings explained so far are not enough. Consequently, we have to define states that will not be modified or must be attendant from others, designated by predicates and facts.

2.4.1 Facts

There are some components that we need to hold along all states. Considering that, we assume some constraints that will always hold.

The initiation (*Init*), that corresponds to the first state of the model. This initiation is very important to keep the system consistent, and aggregate with the predicates that we also define, we can ensure the model verification.

For every state transition, we need to ensure that the model is valid to some invariants. Depending the states of some the components, we want to verify some properties that predicates contain.

```
    fact Init {
        PilotInterface.button.first = Up
        LandingSet.landingset.first = Lock
        Door.door.first = Close
        Hydraulic.hydraulic.first = Extend
        DigitalSystem.anomaly.first = Undetect
        AnalogicSystem.manual.first = Inactive
        PilotInterface.flightSensor.first = Off
    }

    fact Trans {
        all s : State, s' : s.next {
            oneState [s']
            moving [s, s'] or halt [s, s'] or manualsystem [s, s'] or
anomalyDetector [s, s']
        }
    }
```

2.4.2 Predicates

This predicates are the invariants of the model. Those invariants will keep the correct validation of the model, and will allow that the we can limit the next state of the model.

```
    pred oneState [s : State] {
        one button.s
        one landingset.s
        one door.s
        one hydraulic.s
        one anomaly.s
        one manual.s
        one flightSensor.s
    }

    pred moving [s, s' : State] {
        AnalogicSystem.manual.s = Inactive
        DigitalSystem.anomaly.s = Undetect
        LandingSet.landingset.s = Lock
        Door.door.s = Close

        PilotInterface.button.s' = PilotInterface.button.s
        LandingSet.landingset.s' != LandingSet.landingset.s
        Door.door.s' != Door.door.s
        Hydraulic.hydraulic.s' = Hydraulic.hydraulic.s

        DigitalSystem.anomaly.s' = DigitalSystem.anomaly.s
        AnalogicSystem.manual.s' = AnalogicSystem.manual.s
    }

    pred halt [s, s' : State] {
        DigitalSystem.anomaly.s = Undetect
        LandingSet.landingset.s = Unlock
        Door.door.s = Open

        PilotInterface.button.s' != PilotInterface.button.s
        LandingSet.landingset.s' != LandingSet.landingset.s
        Door.door.s' != Door.door.s
        Hydraulic.hydraulic.s' != Hydraulic.hydraulic.s

        PilotInterface.flightSensor.s' != PilotInterface.flightSensor.s
    }

    pred manualsystem [s, s' : State] {
        AnalogicSystem.manual.s = Active
        DigitalSystem.anomaly.s = Detect
        LandingSet.landingset.s = Lock
        Door.door.s = Close

        PilotInterface.button.s' = PilotInterface.button.s
```

```
        LandingSet.landingset.s' != LandingSet.landingset.s
        Door.door.s' != Door.door.s
        Hydraulic.hydraulic.s' = Hydraulic.hydraulic.s
    }

    pred anomalyDetector [s, s' : State] {
        DigitalSystem.anomaly.s = Detect
        AnalogicSystem.manual.s = Inactive

        AnalogicSystem.manual.s' != AnalogicSystem.manual.s
    }
```

2.4.3 System Evolution and Execution

With the purpose to order the system evolution along all of program execution, we implement the concept of *State*. Therewith, we can ensure that all the states modelled respect the characteristics necessary and developed in order to accomplish a valid simulation.

For the execution we use the next 'run' command:

run for 3 but X State, 1 PilotInterface, 1 LandingSet, 1 Door, 1 Hydraulic, 1 DigitalSystem, 1 AnalogicSystem

On *3.2 Model Instances* attachment it is possible to see the instances generated.

3 Attachments

3.1 Alloy Model

```
open util/ordering[State]
sig State {}


sig PilotInterface {
    button : Bstate -> State,
    control : one DigitalSystem,
    trigger : one AnalogicSystem,

    flightSensor : OFstate -> State
}

abstract sig Bstate {}

one sig Up, Down extends Bstate {}

abstract sig OFstate {}

one sig On, Off extends OFstate {}

sig DigitalSystem {
    anomaly : DSstate -> State,
    emergency : one AnalogicSystem,
    handle : one LandingSet
}

abstract sig DSstate {}

one sig Detect, Undetect extends DSstate {}

sig LandingSet {
    landingset : Lstate -> State,
    valvesDoor : one Door,
    valvesHydraulic : one Hydraulic
}

abstract sig Lstate {}

one sig Lock, Unlock extends Lstate {}
```

```
    abstract sig Door {
        door : set Dstate -> State
    }

    abstract sig Dstate {}

    one sig Open, Close extends Dstate {}

    abstract sig Hydraulic {
        hydraulic : set Hstate -> State
    }

    abstract sig Hstate {}

    one sig Extend, Retract extends Hstate {}

    abstract sig AnalogicSystem {
        manual : ASstate -> State
    }

    abstract sig ASstate {}

    one sig Active, Inactive extends ASstate {}


    fact Init {
        PilotInterface.button.first = Up
        LandingSet.landingset.first = Lock
        Door.door.first = Close
        Hydraulic.hydraulic.first = Extend
        DigitalSystem.anomaly.first = Undetect
        AnalogicSystem.manual.first = Inactive
        PilotInterface.flightSensor.first = Off
    }

    fact Trans {
        all s : State, s' : s.next {
         oneState [s']
         moving [s, s'] or halt [s, s'] or manualsystem [s, s'] or
anomalyDetector [s, s']
        }
    }
```

```
pred oneState [s : State] {
    one button.s
    one landingset.s
    one door.s
    one hydraulic.s
    one anomaly.s
    one manual.s
    one flightSensor.s
}

pred moving [s, s' : State] {
    AnalogicSystem.manual.s = Inactive
    DigitalSystem.anomaly.s = Undetect
    LandingSet.landingset.s = Lock
    Door.door.s = Close

    PilotInterface.button.s' = PilotInterface.button.s
    LandingSet.landingset.s' != LandingSet.landingset.s
    Door.door.s' != Door.door.s
    Hydraulic.hydraulic.s' = Hydraulic.hydraulic.s

    DigitalSystem.anomaly.s' = DigitalSystem.anomaly.s
    AnalogicSystem.manual.s' = AnalogicSystem.manual.s
}

pred halt [s, s' : State] {
    DigitalSystem.anomaly.s = Undetect
    LandingSet.landingset.s = Unlock
    Door.door.s = Open
    PilotInterface.button.s' != PilotInterface.button.s
    LandingSet.landingset.s' != LandingSet.landingset.s
    Door.door.s' != Door.door.s
    Hydraulic.hydraulic.s' != Hydraulic.hydraulic.s
    PilotInterface.flightSensor.s' != PilotInterface.flightSensor.s
}

pred manualsystem [s, s' : State] {
    AnalogicSystem.manual.s = Active
    DigitalSystem.anomaly.s = Detect
    LandingSet.landingset.s = Lock
    Door.door.s = Close
```

```
        PilotInterface.button.s’ = PilotInterface.button.s
        LandingSet.landingset.s’ != LandingSet.landingset.s
        Door.door.s’ != Door.door.s
        Hydraulic.hydraulic.s’ = Hydraulic.hydraulic.s
    }

    pred anomalyDetector [s, s’ : State] {
        DigitalSystem.anomaly.s = Detect
        AnalogicSystem.manual.s = Inactive
        AnalogicSystem.manual.s’ != AnalogicSystem.manual.s
    }


    run {} for 3 but 3 State, 1 PilotInterface, 1 LandingSet,
1 Door, 1 Hydraulic, 1 DigitalSystem, 1 AnalogicSystem
```

3.2 Model Instances

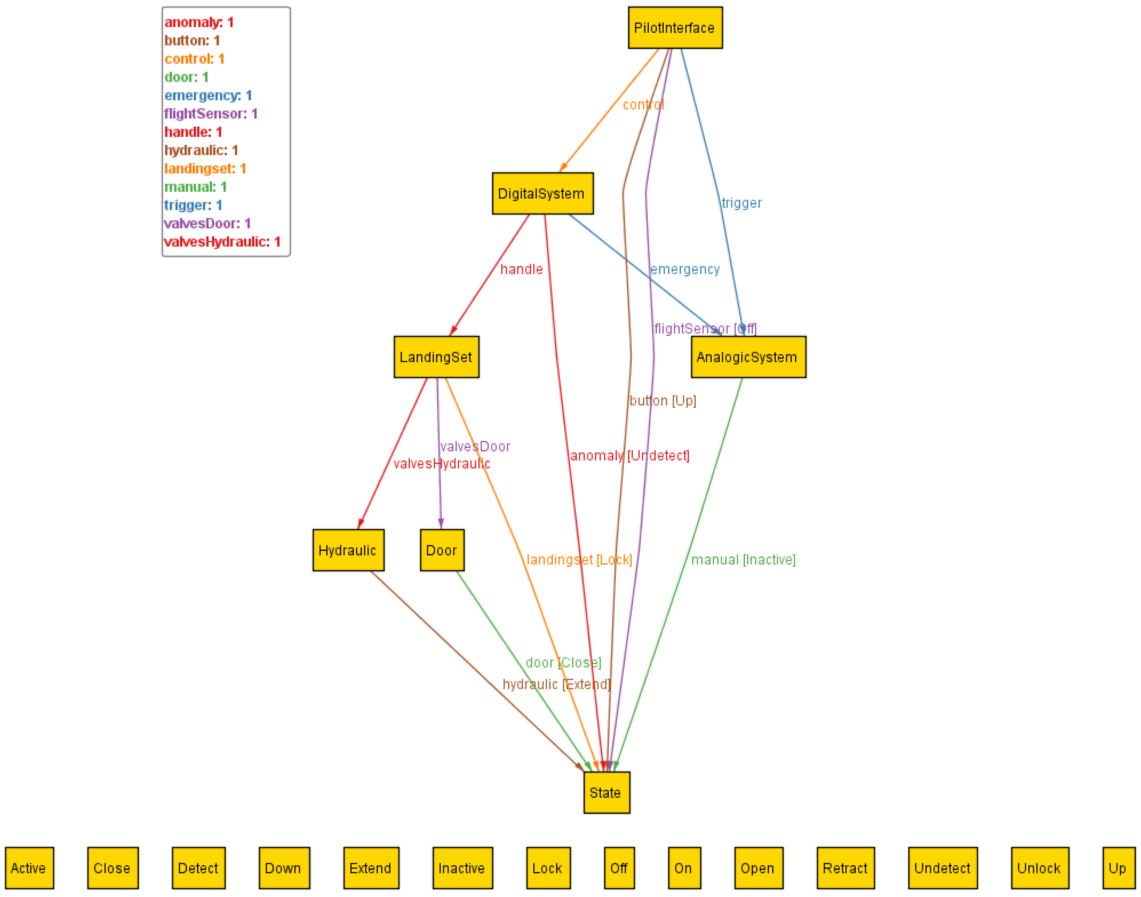

Figure 2: Instance with 1 State

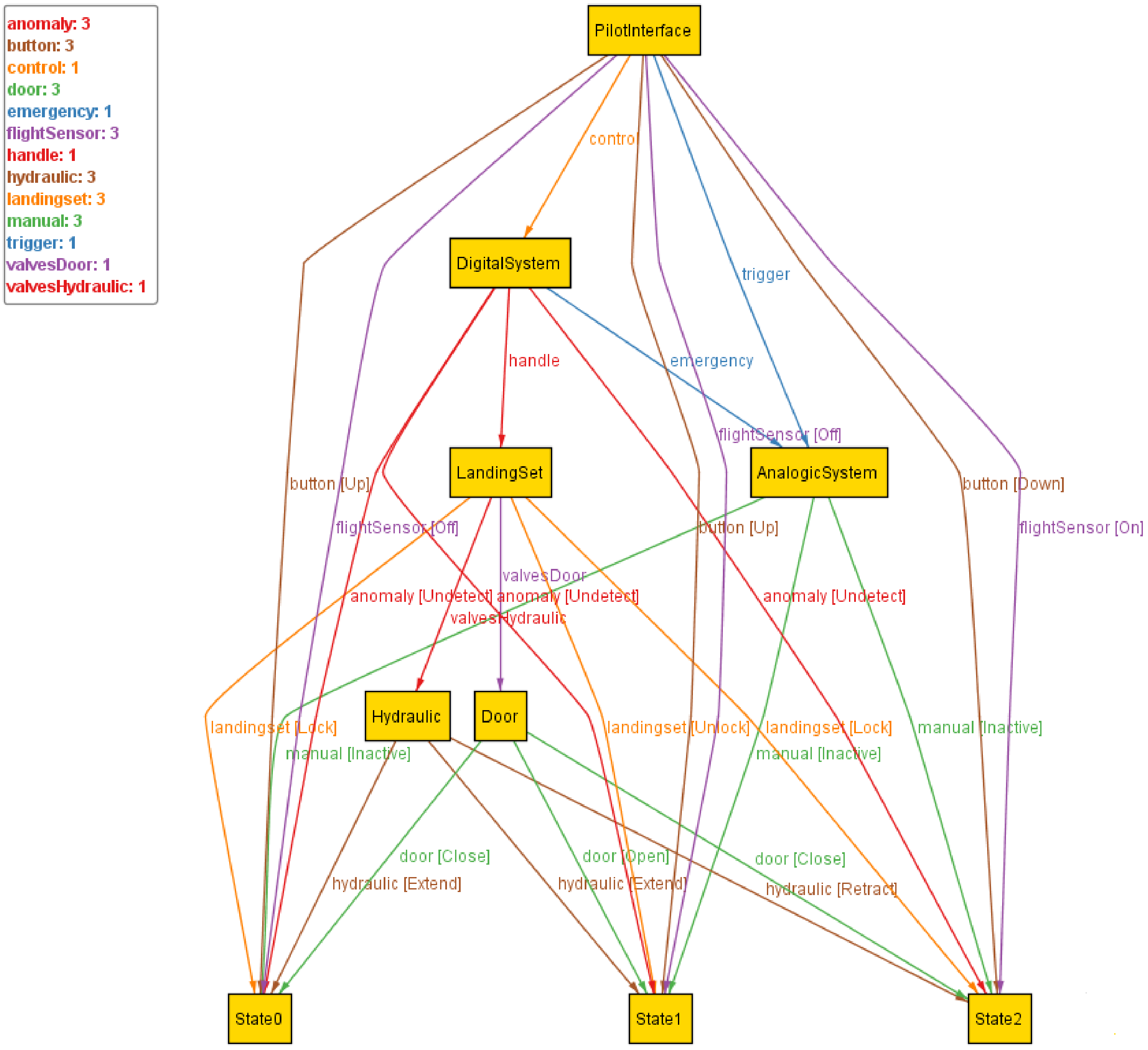

Figure 3: Instance with 3 States

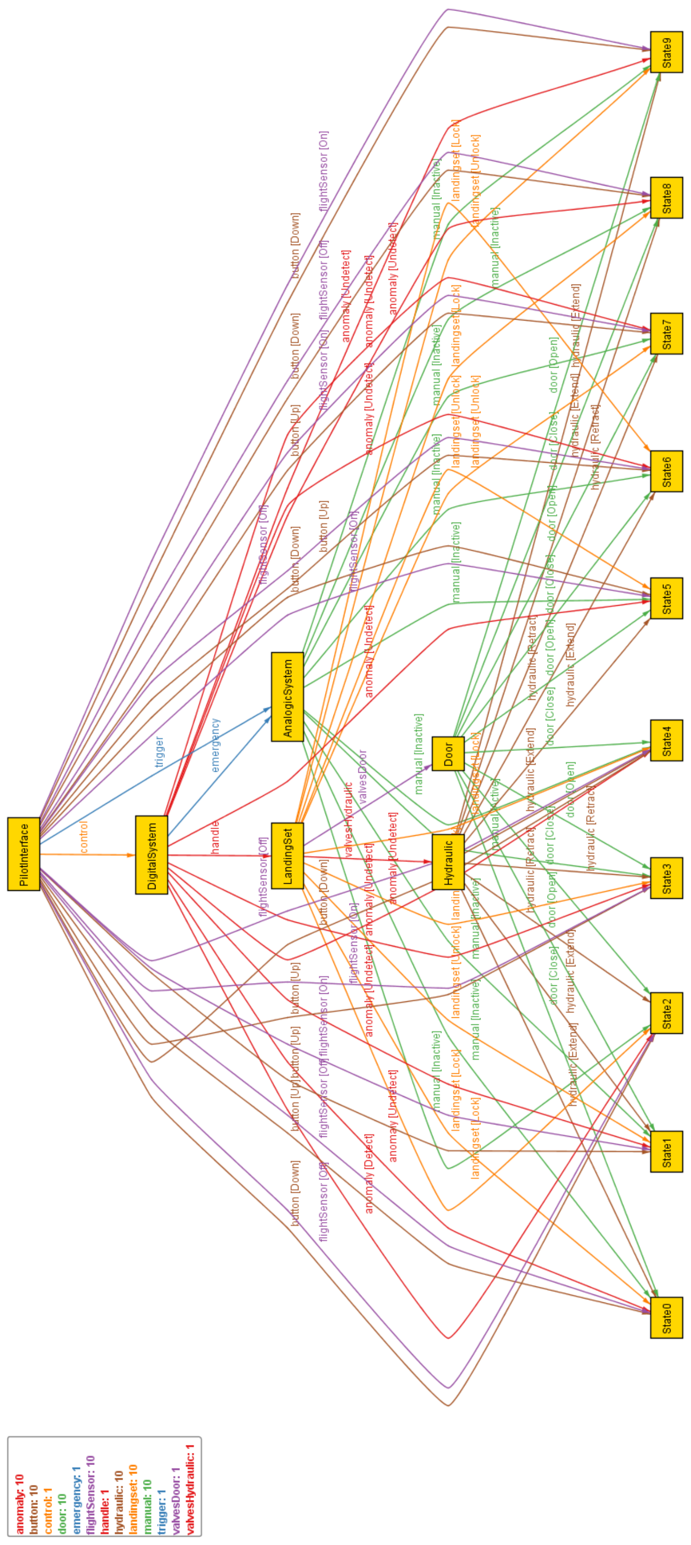

Figure 4: Instance with 10 State