UNIVERSITY of the CORDILLERAS

SYSTEM VALIDATION: AUTOMATA AND BEHAVIOURAL EQUIVALENCE

Creator:
Aben, John Paul
Orsolino, Abigail
Soriano, Ajirah Ramielle

CITCS-2H

Instructor:
Sir Jeffrey, Ingosan

System Validation: Automata and Behavioural Equivalence

Student Name: ____________________________________

CITCS Department

University of the Cordilleras

Syllabus

Part I

Behavioural Modelling

- Introduction of behavioural Modelling
- The definition ofan Automaton
- An Automaton as a model for behaviour
- Non-Deterministic

Part II.a

Behavioural Equivalence

- What is Behavioural Equivalence?
- Strong Bismulation
- Trace Equivalence
- The Internal Action
- Branching Bisimulation
- Rooted Branching Bisimulation

Part II.b

Behavioural Equivalence

- The Alternating bit Protocol
- Divergence Preserving Branching Bisimulation
- Weak Bisimulation
- Weak Trace Equivalence
- Language, Failure, and Completed Trace Equivalence
- When to use which Behaviour?
- Transition Systems with data, time and probabilities

PART I

BEHAVIOURAL MODELLING

System Validation: Automata and Behavioural Equivalence

Objectives:

- ➢ To learn to use the behaviour of a computer as an automaton
- ➢ To understand and define what it means for two behaviors to be equivalent
- ➢ To learn the notion of an internal action and how we can use it to get insight in behaviour

What is System Validation?

- o System Validation is the field that studies the fundamentals of system communication and information processing.
- o It allows automated analysis based on behavioural models of a system to see if a system works correctly, to guarantee that the systems does exactly what it is supposed to do and to prove the absence of errors.
- o It allows to design embedded system behaviour that is structurally sound and as a side effect enforces you to make the behaviour simple and insightful.

What is the essential of studying and understanding Automata and Behavioural Equivalence?

- ❖ Automata will let us to model the behavior of computers and to understand when two of these behaviors are equal or different.
- ❖ It will let us understand and model the behaviour of software-controlled systems with a simplest and basic way using automata or labeled transition system.

This work book will talk about automata and process equivalence, we will show you how you can use Automata to model the behavior of computers, We will also let you understand the importance of system validation, and the terms and functions of how the model behavior of systems works.

Key Concepts

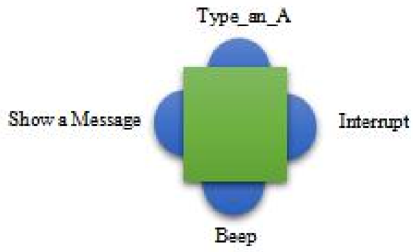
Terms	Definition
Input	A string fed to a machine which the machine will determine whether it is part of the language that the machine was designed for.
Return	The results of running the machine on a given input. Initially, this will either be **accepted** or **rejected**, indicating whether the input string is respectively part of the language or not.
State	A resting place while the machine reads more input, if more input is available. States are typically named.
Start State	It is the state that the machine naturally starts in before it reads any input. It is also known as the *program entry point*.
Accepting State or **Final State**	A set of states which the machine may halt in, provided it has no input left, in order to accept the string as part of the language.
Dead State	A rejecting state that is essentially a dead end. Once the machine enters a dead state, there is no way for it to reach an accepting state, so we already know that the string is going to be rejected.
Transition	A way for a machine to go from one state to another given a symbol from the input. Graphically, a transition is represented as an arrow pointing from one state to another, labeled with the symbol or symbols that it can read in order to move the machine from the state at the tail of the arrow to the tip of the arrow. Transitions may even point back to the same state that they came from, which is called a **loop**.
State machine	A state machine is a mathematical model of computation. It's an abstract concept whereby the machine can have different states, but at a given time fulfills only one of them.
Rejecting State/ Terminating State	Any State in the machine which is not denoted as an accepting state. The string is only rejected if the machine halts in a rejecting state with no more input left/

Automaton as a Model for Behaviour

Objectives:

- To understand the notion of Label Transition system to model behavior.
- To understand the notion of a state and initial state and the important notion of an atomic action.
- To define the use of a transition, using an atomic action from a state to another state.
- To identify and understand the traces of a transition system

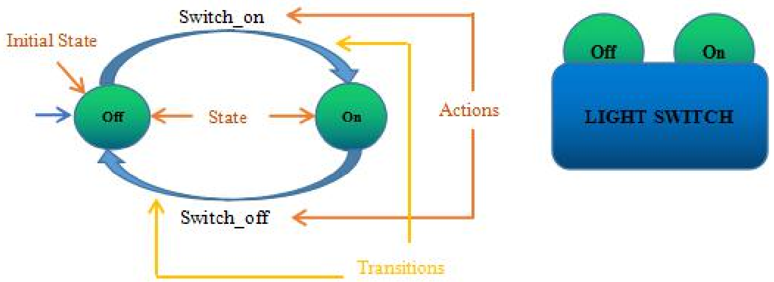
Robin Milner in 1973, is one of the persons who had a very particular ideas how they can model the behavior of an IBM 360 computer, He said that we are all looking at behavior of these systems as functions. Functions from input to output, where in a program takes input and writes the output, but that is not the correct way of looking at systems. What Milner said was, systems are always interacting with each other, and input comes, output comes, it goes all together.



He said that a good model is just like this box with buttons on it, and these buttons represents the actions that you can press, and they stand for something that happens. For instance, you are using a computer, then it could be that you will be typing an A on a keyboard and that is an action that can happen. And, there could be an interruption happen, also, Old computers could still say beep as an output, and they could write messages on a screen. These are all actions that can happen, and they are input actions and output actions, and they really can happen in any arbitrary sequence.

Now if you have this kind of behaviour, the first idea is that we take all these actions (*Actions are atomic events that have zero duration.)* and reconsider them as being atomic. Actions are events that really happen in an instance of time, and we see it as an atomic event.

What is then an automaton? To further understand what an automaton is, we will have an example: Light Switch



This light switch can be switched on and switched off, and there are two actions, namely, switch on and switch off. We model it with these two things that are called states, and the whole light can be in a state off and can be in state on, this picture or process are called labeled transition system or state machine or an automata.

The state off is the initial state wherein the whole system starts, initially the light is off and now using transitions we can move from state to state when a particular action is happening.When the light is off we will do then an action transition switch on to move on to another state which is the state on, and if you like to switch it off then you do an action switch off and you'll be then moved to another state which is the state off. And with these actions and transitions, we can now define the trace of the system.
Trace = {Switch_on, Switch_off, Switch_on, Switch_off, Switch_on, Switch_off,…}

A trace is simply switch sequence of actions and that represents the actions you can do with the system.

Labeled Transition System

objectives:

- To define a transition system using set theory
- To define and understand what unreachable states and deadlock means in a transition system
- To understand how finite or infinite numbers of actions and transitions work.
- To formalize a transition System

What is Labeled Transition System / Automaton?

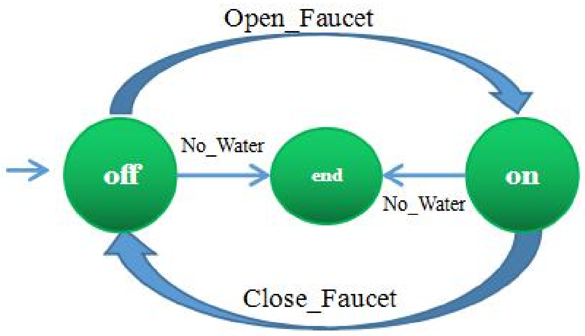
Automaton or A **labeled transition system** consists of a set of states plus a **transition** relation on states. Each **transition** is **labeled** with an action. It is a natural and straightforward way to describe the **behavior** of a concurrent system. Labeled transition system is also described as a five Tuple which represented by the following:

(S, Act, ->, S_o, T)

S	It stands for the set of State, these set can either be finite or infinite set.
Act	It represents the Actions; these actions can also be finite or infinite.
$\rightarrow$	This arrow represents the transition relation or transition function mapping, this transition relation is a guide or a process that will help you on how you get from one state to another state via an action
S_o	Initial State
T	It represents the set of terminating states

To further understand how the labeled transition works, here are some examples:

I. First example: Water Cycle in a faucet.

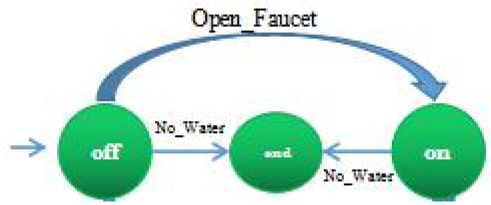


As you can see on this example, there are 3 states, the On, Off and an end state which is a representation of a Terminating State. And to formalize this transition system it would look like this:

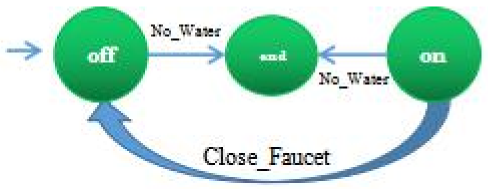
S	={off, on, TS}
Act	={Open_Faucet, Close_Faucet, No_Water}
→	={(off, Open_Faucet, On),(off, No_Water, end), (On, Close_Faucet, off), (On, No_Water, end)}
So	={off}
T	={end}

- ➢ The set of states are the states off, on, and end.
- ➢ The set of actions are Open_Faucet, Close_Faucet, and No_Water.
- ➢ The transitions consist of Four, and these are the following:
    - o Transition from state off with the open faucet action to the state on.
    - o Next is a state off with the No_Water action to the state end.
    - o There's also a transition from state on, with the close faucet action to the state off.
    - o And a state on with the No_Water action to the state end.
- ➢ The Initial state is off.
- ➢ And the terminating state is end.

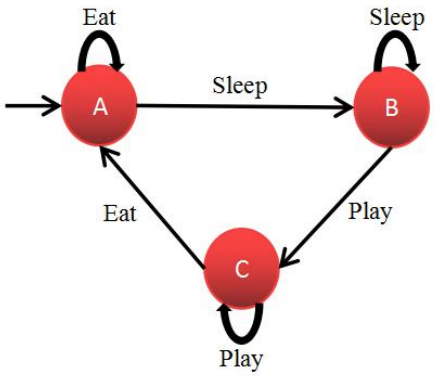
This labeled transition picture is based on a real-life process of using a faucet. Where in if you open a faucet there are two possibilities either the water will run out or not:



And on the other hand, once the water came out on the faucet there will also be a possibility that the water will run out, or you can just close the faucet and considered it off.



II. Here is another example, and now it is about a transition system with an infinite number of states:



If we formalized that Labeled transition it would be like this:

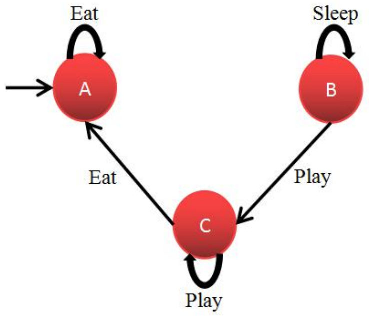
S	= {A, B, C}
Act	= {Eat, Sleep, Play}
→	= {(A, Eat, A), (A, Sleep, B), (A, Sleep, B, Play, C), (A,Eat, A,Eat, A), (A, Sleep, B, Sleep, B), …..)}
So	= {A}
T	= {}

- The set of states are the states A, B, and C.
- The set of actions are Eat, Sleep, and Play.
- The transitions consist of an infinite transitions where in; State A can go with an infinite action of Eat, and State B can also go with an infinite action of Sleep, and State C can also go with an infinite action of play, or it can also be an infinite transition of state A with the action of Eat to the State B with the action of Sleep to the state C with the action of Play and back to the state A and so on.
- The Initial state is A.
- And the terminating state is an empty set.

But what do you think will happen if we remove some transitions from the transition system?

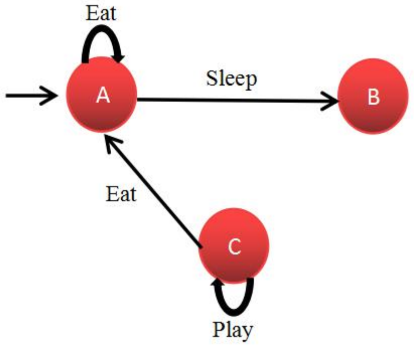
To further understand the Labeled Transition System, we will also define and understand first what is an unreachable and a deadlocked state

II.a Example:



In this Example of transition system, if you compare it to the 2nd example, the action transition of Sleep is gone, which means that the B and C states are now unreachable, and the transition system will just be looping the Eat action. This is quite common in transition systems, when it comes to a larger set of transition system, those unreachable states are generally the states that are not so interesting. And only a small fragment is often reachable.

II.b



On this next example we remove the loop of Sleep action and the transition action of Play, and because of that then you see that there is no more outgoing transition from state B. Where in If you are in state B you are stuck there. This is called a deadlock or a deadlock state. And if a system has a deadlock state this is generally undesirable.

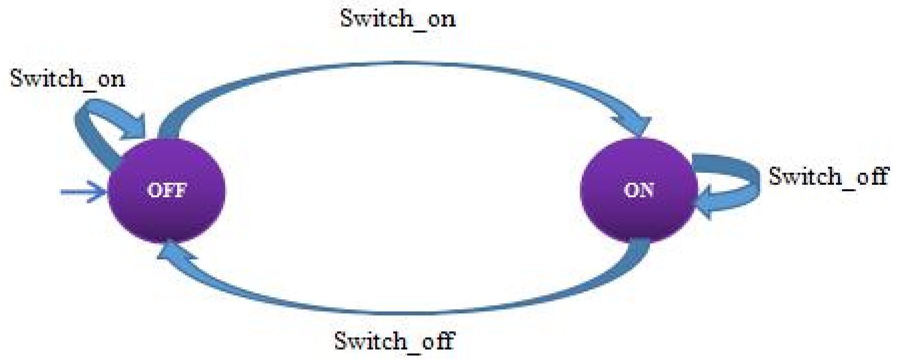
Non-Deterministic Behaviour

Objectives:

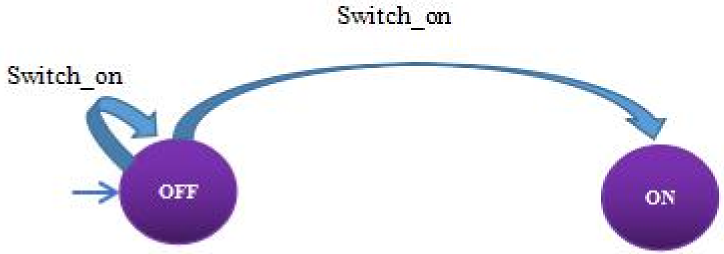
- How can we use non-deterministic behaviour to design behaviour leaving some freedom.
- and compactly model complex behaviour (Milner's weather freedom)

Machines are categorized into different types. Each type behaves radically different from other types and tend to appear graphically different (especially the labels for the transitions). There are two types of Labeled Transition System and these are; Finite Automata (FA), and the Deterministic Finite Automata (DFA). It is called a Finite Automata because we know that given a finite input, the machine will execute in reasonable finite time and give us a result. This is called **Halting**. It is Deterministic if every state has exactly one transition per symbol, and there are no ambiguities as to which state to go to on a given symbol of input. The Second type is the Non-deterministic Finite Automata (NFA).

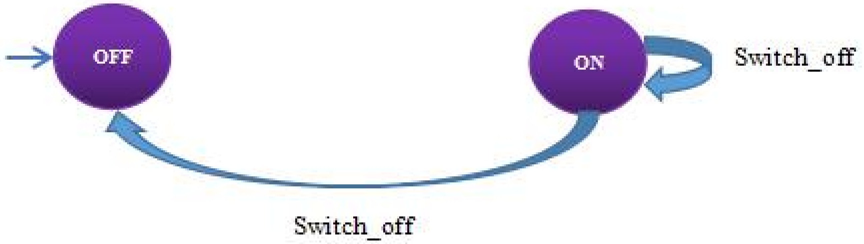
We say that a state is non-deterministic if, and only if, it has at least two outgoing transitions within the same label. Example:



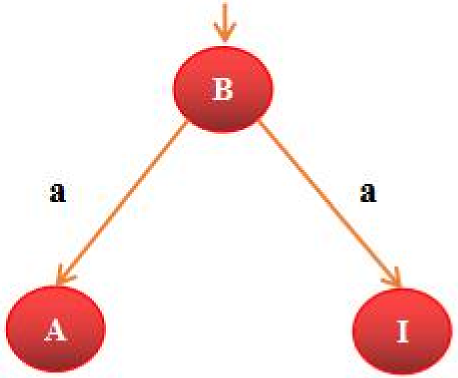
That example is a non-deterministic because the Transition System of the off state has two outgoing transitions going to different states with the same label.



Same with the on state

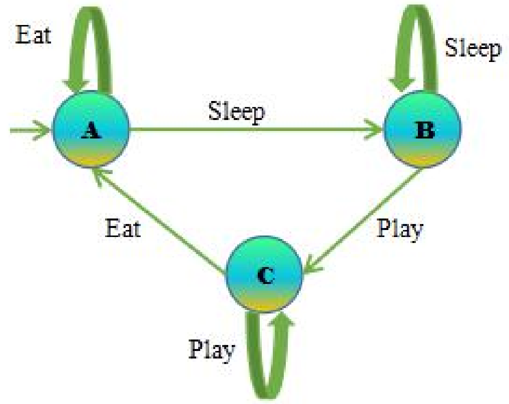


To make it simpler here is a simple representation of a non-deterministic behaviour:

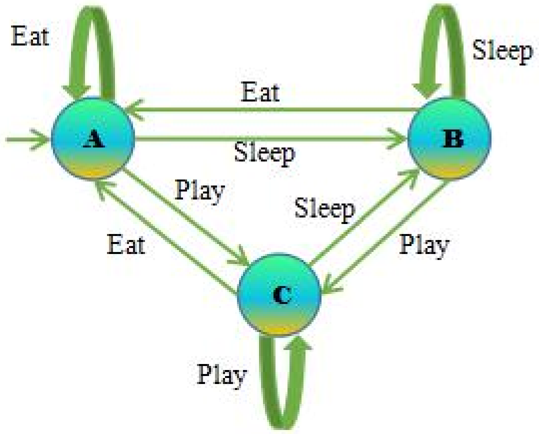


What is now the reason for this non-determination? Milner already calls it the *weather conditions*. He said that 'if you look at switch, the switch may be influenced by the weather and the weather may determine whether the switch will work properly or not'.

What he says is, by using non-determinism, in a very compact and effective way, we can model much more complex mechanisms, for example:



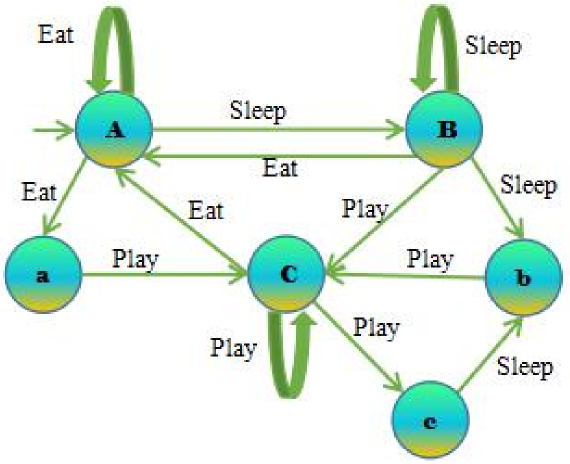
Here is an example of a deterministic behaviour, and it can also be like this:



So in that transition system there are a lot of possible action you can do, either you eat twice before you sleep or you eat first before you sleep and sleep again and again before you play or you can just eat after that you sleep and after that you play nonstop, and so on.

But what if you don't like this kind of behaviour? Because you don't want to sleep, you only want to eat and play, or you just want to eat and sleep.

Using non-determinism, we will redesign that behaviour according on our desired behaviour, and it would look like this:



As you notice the transition System become a little bit complicated. But in this behavior, you can now do any action based on what you wanted to do.

With the use of non-deterministic bahaviour you can easily manipulate or design the behaviour of your system, it might be a little complicated, but the output is satisfiable.

ACTIVITY

Based on what you've learned, answer the following exercise:

I. Matching type

Instruction: Match the definition listed on set B that corresponds on the terms in Set A

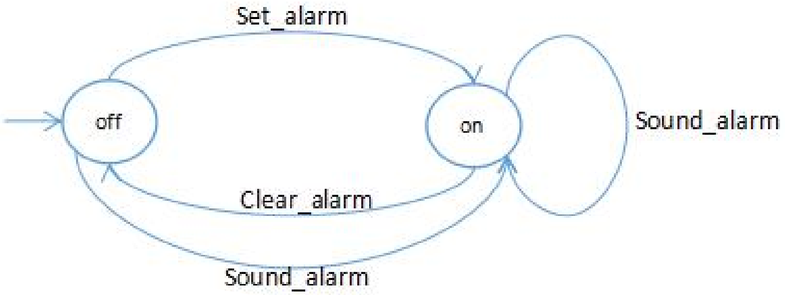
Answers	Set A	Set B
1.	Terminating State	a) It is a Binary relation between state transition systems, associating systems that behave in the same way.
2.	So	b) A rejecting state that is essentially a dead end.
3.	Deadlock	c) A set of states which the machine may halt in, provided it has no input left
4.	Act	d) A relation that will help you on how you get from one state to another state via an action
5.	Automaton	e) Any State in the machine which is not denoted as an accepting state.
6.	Transitions	f) A type of a infinite automaton that can move to any combination of the states in the machine and has a finite number of states.
7.	State	g) these set can either be finite or infinite set.
8.	NDFA	h) It is an abstract self-propelled computing device which follows a predetermined sequence of operations automatically.
9.	Final State	i) It is the state that the machine naturally starts in before it reads any input.
10.	Bisimulation	j) It represents the Actions; these actions can also be finite or infinite.

II. Identification

Instruction: Identify the states, initial state/s, actions, transitions, and terminating states of the given Labeled Transition Systems. Write your answers on the box provided.

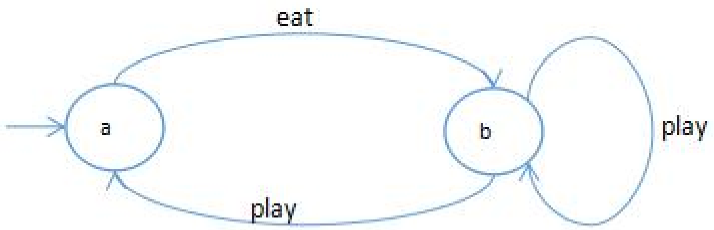
A.

S	
So	
Act	
→	
T	



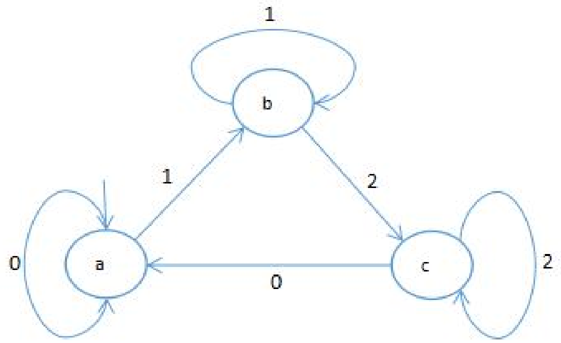
B.

S	
So	
Act	
→	
T	



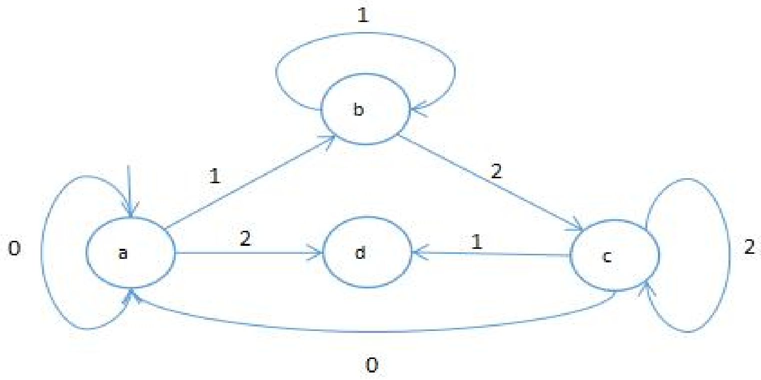
C.

S	
So	
Act	
→	
T	



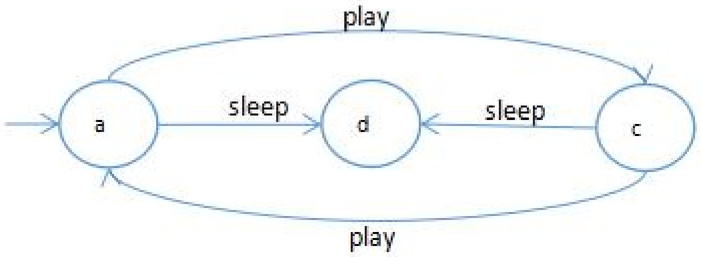
D.

S	
So	
Act	
→	
T	



E.

S	
So	
Act	
→	
T	



III. True or False

Instruction: Write T if the statement is True, write F if the statement is False.

______1. Actions are events that really happen in an instance of time, and we see it as an atomic event.
_____2. Robert Miller said that systems are always interacting with each other, and input comes, output comes, and it goes all together.
_____3. Labeled transition system is also described as a five Tuple.
_____4.The transition from a state going to a single particular next state for each input symbol is called NDFA.
_____5. The transition from a state can be to multiple next states for each input symbol is called DFA.

PART II.A

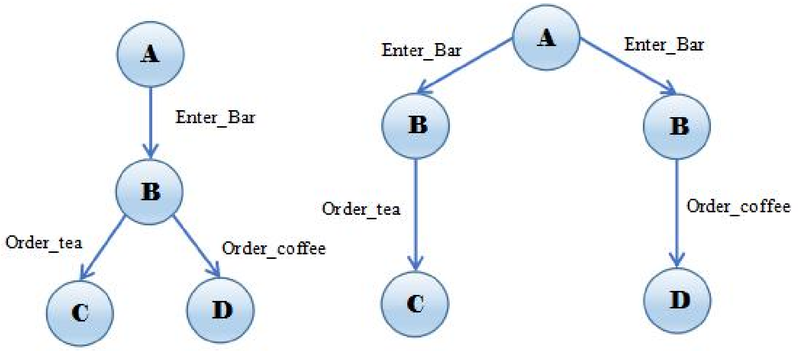
BEHAVIOURAL EQUIVALENCE

Behavioural Equivalence

In this section we will be exploring the question, when are two transition systems equivalent?

We have here a two transition systems that are quite similar, but the question is, are they also the same from a behavioural perspective? the answer is, it depends on our interactions in the system. In some cases, they are not the same, and in some cases we would like to consider them as behavioral equivalent.

To further understand here is the example:



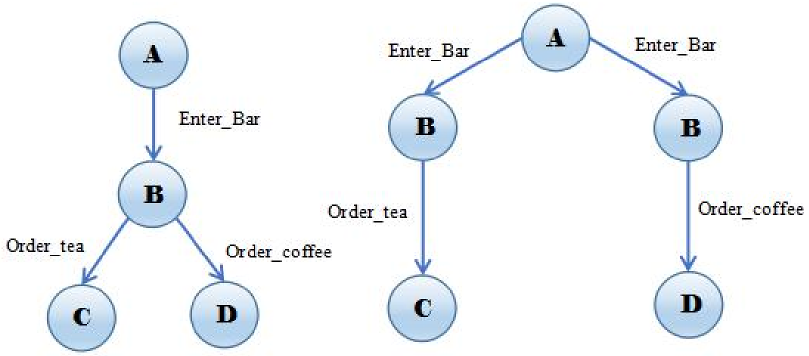
states = { a, b, c, d}
Act= {enter_bar, order_tea, order_coffee}

On the first transition you will be entering a bar, and you can choose either to order tea or to order coffee, and on the other hand there is a transition system that will show us a certain process where in you either enter_bar, then just order_tea or enter_bar, and just order_coffee.

The first transition, is where we can make a choice whether to order tea, or order coffee however on the second transition system, once you enter the bar, you can either only order a coffee or a tea, for example you don’t like a tea but you enter the bar going to state C then you don’t have a choice but to order tea.

In this case we have an interaction with the system, on the first one we can actually make a choice, however on the second one you don’t have a choice, so basically we cannot consider these two transitions systems as the same. And the notion that's belongs to it is bisimulation equivalent. However These two transition systems are not bisimulation equivalent.

Another example:



Same scenario, but now you are not the customer, you are just a passer-by in that restaurant, suppose we are looking at the same behaviors, but now as if we were just watching another customer ordering their drinks. We do not have interaction, we just observe what can happen. Example a customer enter a bar and he or she has a choice whether to order coffee or tea, at last he or she ordered a coffee, then another customer enter a bar and he or she directly ordered a coffee, and because you’re just an observer, you will think that it is a the behaviour wherein that customer ordered coffee, And in that case, we would like to consider these two as equivalent. And we call this trace equivalence

.

The person who worked on all these equivalences and actually made a big lattice, the van Glabbeek lattice, is Rob van Glabbeek. He makes the following observation, namely every way of interacting with the system induces a new equivalence. And there are literally hundreds of equivalences depending on how you look at a system. Fortunately, bisimulation is a safe choice always, because it preserves every reasonable behavioral property that we can think of.

BRANCHING BISIMULATION

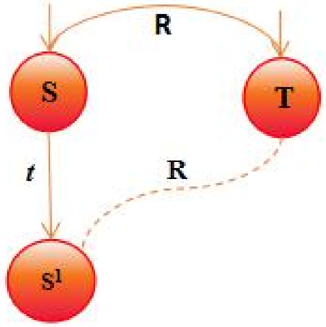
Objectives:

- To identify how transition systems can be prove or disprove to be branching bisimilar
- To understand clearly what branching bisimulation

Branching bisimulation is similar with strong bisimulation, The only difference is that we take internal actions into account.

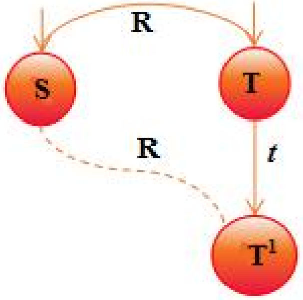
Here are some Examples:

I. Example:



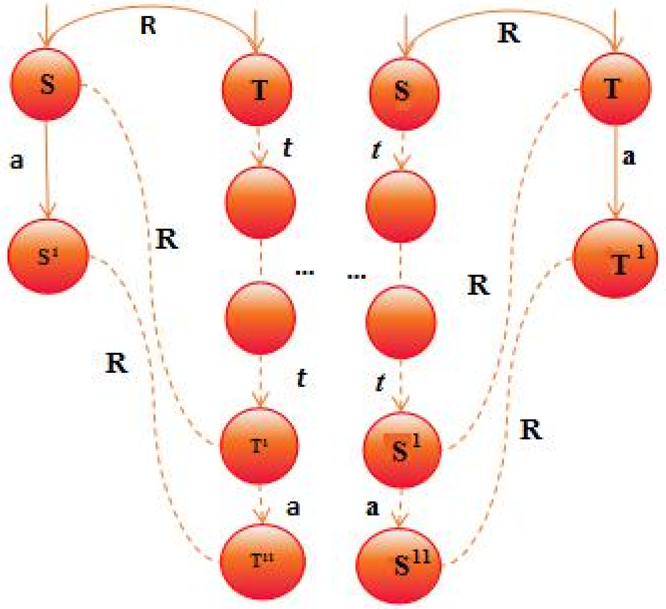
This is a Branching bisimulation because both bisimulation has the same traces, wherein we relate the two initial states, and if we trace their action the output will be the same, just like the first transition system wherein after the initial state S we have a tau which means no action and then we move to S prime, it is the same to the second LTS where in it has only initial states and no action has been done.

(Symmetry case)

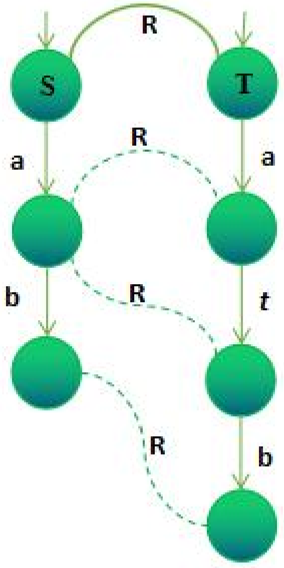


In these case the relation should be symmetric. And that means that the other side, the symmetric case, should hold. Suppose we have, again, two states. The right hand side can do a tau step. And then the tau step can be mimic by simply doing nothing at the left-hand side, and we must relate the S state to the state T.

II. Another example of a branching bisimilar with symmetry case.

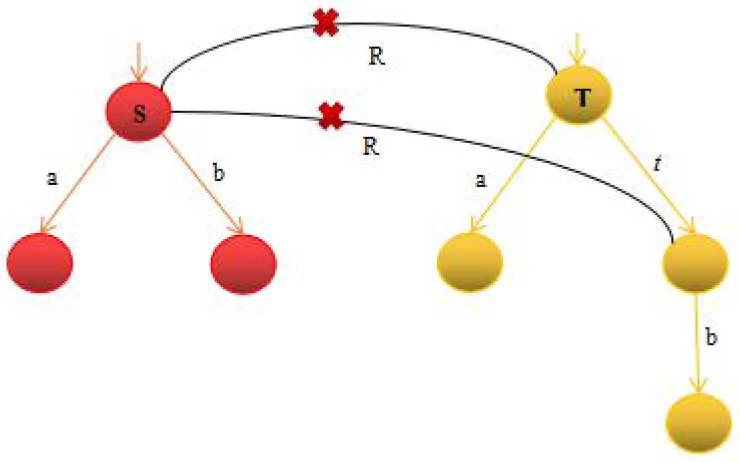


III. Example:



We relate the two initial states. And then we mimic the steps at both sides. Therefore if you do an a at left, you should be able to do an a at the right and vice versa. This means that these two states would also be related. And now, if you do the tau at the right, then we mimic it by simply doing nothing on the left, and we relates these two states, and that is allowed in branching bisimulation. And if you look at the last two related states, then the b step can be simple mimicked by the other side, and the end states are related again. And these are branching bisimilar.

IV. Here is an example of a not branching bisimulation:



We do first this initial step, and relate to the initial states. If you have a tau state, the tau translation at the right hand side, then you can only mimic that by staying in the initial state at the left hand side. If the first two are related, then these two states must also be related. But now we can observe a problem, namely in the states at the left. We see that a action can be done and this cannot be done at the right, the state in the middle, right hand side. So there is no a action, and that means that the two letter states, cannot be related. And that means that they had to be related if the initial states were related. But in this case they are not related therefore the initial states can also not be related. This way we can figure out that these two are not bisimilar.

Strong Bisimulation

In this topic we will be dealing with one of the important notions, which is what we call the notions of bisimulation.

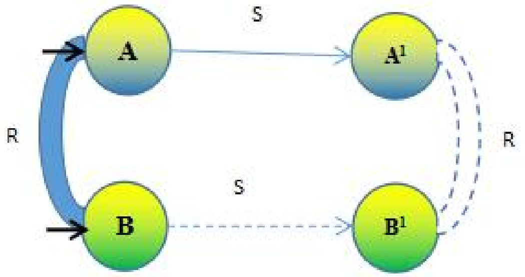
What is bisimulation?

Bisimulation is a binary relation between state transition systems, associating systems that behave in the same way in the sense that one system simulates the other and vice versa. Intuitively two systems are bisimilar if they match each other's move or traces.

What is then a Strong Bisimulation??

Strong bisimulation is an equivalence relation that partitions the set of states in such a way that the set of actions that can be executed to reach some class is the same for every two states in a class. In addition, the termination behavior of two related states must be the same.

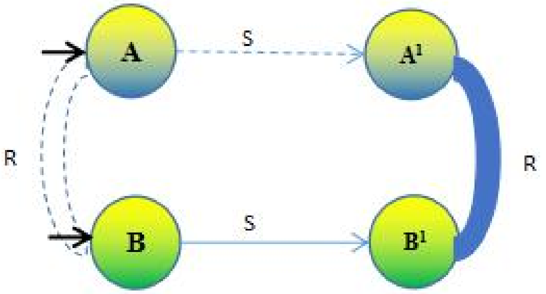
I. Example:



In this example you can see that we have 2 different transition, to identify if this two transition is a strong bisimulation we're going to compare each state and actions to one another.

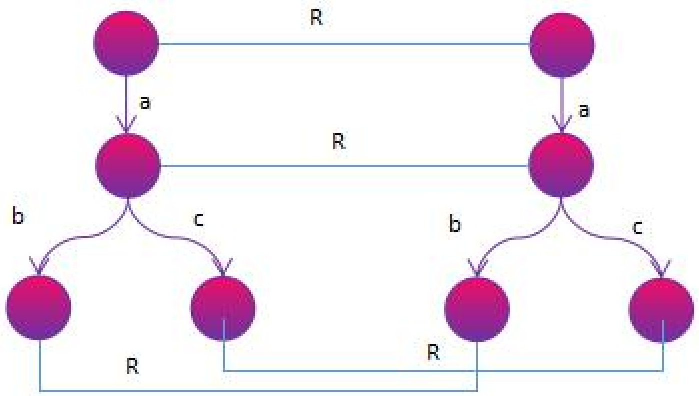
Firstly, we say that state A and state B are related because we are comparing this 2 transition and A and B are their Start state, Second, as you can see state A has a transition s going to A prime, what we're going to do is to compare the action in transition system, A to A prime and B to B prime. Both has actually the same action transition so were going to represent the line of B to B prime in a dashed line, and because both transition systems are the same then we can say that the two Transition system also relates to each other.

The definition of bisimulation also obliges you to check the other side. And it would look like this:



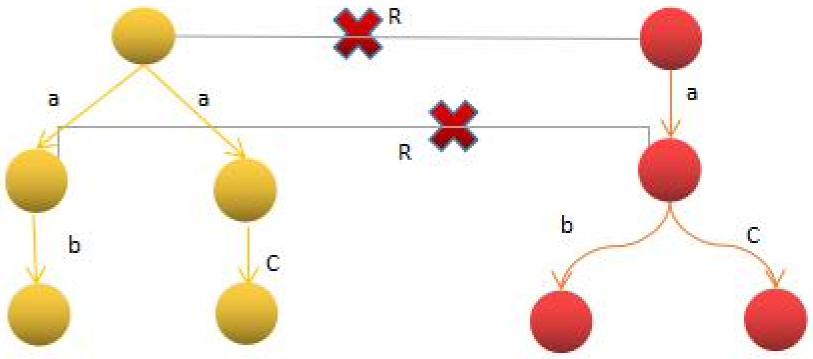
Now when are two states bisimilar? If we have two states S and T and we can find a bisimulation relation R that relates them, then we say that these two states are bisimilar. And if they have two transition systems, we say that the two transition systems are related if the initial states are related.

II. Example:



What we have to show is that if there is a bisimulation relation that relates the these two transition system. Just on the picture above wherein we extend this transition relation over all transitions of overall states and see that it satisfies the properties of bisimulation. So, if we have an a step at the left, then it should be possible to do an a step at the right, and the end point should also be related..

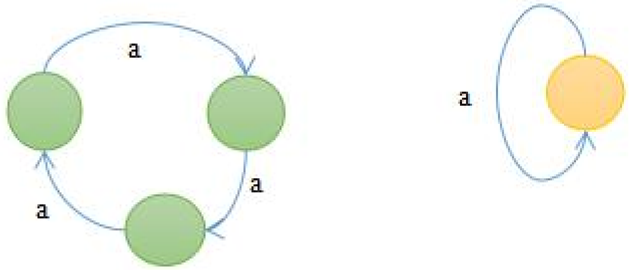
III. Example:



Here is an example of not related transition system. As you can see its possible that the initial states of both transition system can be related but on the second states going to b action can also be related but when you go to the c action you cant say that it cant be related, because the other one contain 2 actions however the other one contain only a specific action, therefore these two transition system cant be related.

Property of a bisimulation equivalence:
Every transition system that you have has a unique minimal transition system, that belongs to it.
And If you have a big transition system, we calculate as minimal transition system and then we store the behavior in this minimal object.

IV. Example



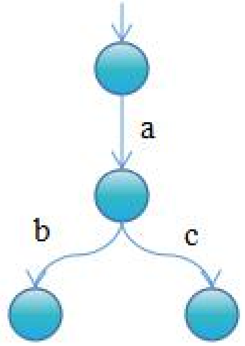
Here we have a transition system that can do three a's and then repeats itself again. And what you can imagine, is this transition system can do typical human behavior of doing an infinite number of a's. On the other hand, there is a much simpler transition system that can also do an infinite number of a's. And this is actually the unique, minimal transition system that is bisimilar to the transition system at the left.

TRACE EQUIVALENT

Objectives:

- To define what it means for two transition systems to be trace equivalents.

Trace Equivalence:



Firstly, now we need to know what a trace of a system is; trace is essentially a sequence of actions that can be executed and here we will write down the set of traces of this process.

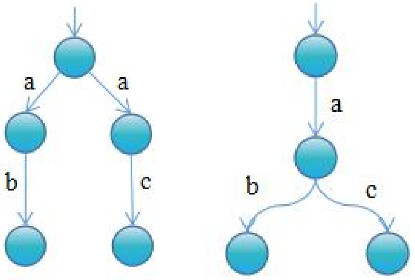
a) The first trace that you can do is simply the empty trace, the trace where you do no action at all and that's generally written as an epsilon or an empty set. Trace: {{}}

b) Second we trace the initial state. Trace: {{}, a}

c) Third we then trace the action wherein based on the transition system, we can make an action a going b and a going to c, if we trace it would be like this: Trace: {{}, a, a*b, a*c}

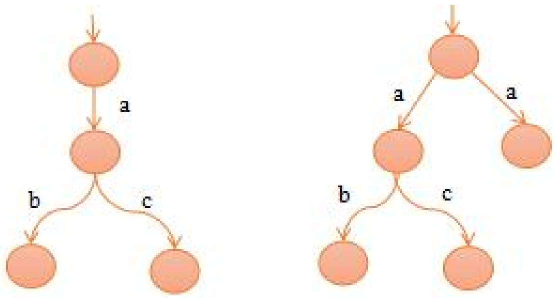
But When are two transition systems equivalent? You can say that the two transition systems are equivalent if the set of traces of it are the same.

I. Example:



Here we have two different structure of a transition system but if we trace there action they actually has the same traces: they can both have a empty set and they also has the same initial state, both also can do an action from a to b and a to c.

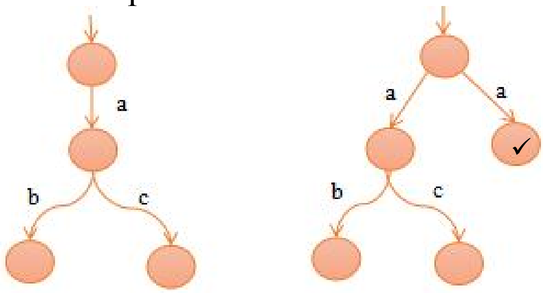
II. Example:



These two transition are another example of a trace equivalent, because it may look like a different transitions but if we trace it, the traces are similar to each other.

The trace of the first transition system are: {{}, a, a*b, a*c}
And on the second transition the traces can be first an empty set as well, then it has a choices whether to go and stop at a which is a deadlock state or it is also possible to go on a state and have an action a to b, and a to c. so if formalize it will become like this: {{}, a, a*b, a*c} Which means that those transition systems are equivalent.

III. Example



Same example from the second example but the difference here is that we put a check mark on one of the deadlock state on the second transition system. If we trace it, the first transition system will have a: trace: {{}, a, a*b, a*c} and the second transition state will have a trace of : {{}, a,a* ✓ , a*b, a*c}
And because of that it wont consider it as a trace equivalent. Therefore the third example is not a trace equivalent.

Things to remember:

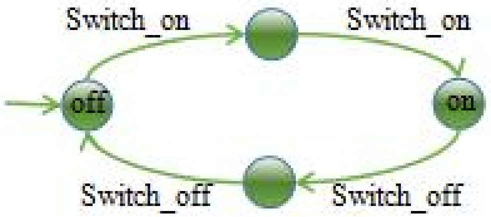
- If two states are bisimilar, then they are also trace equivalent.
- If we have two deterministic transition systems and we would like to know whether they are the same, then it's useful to know that bisimulation and trace equivalence coincides.

Conceptually trace equivalents are very easy and much easier than the definition of bisimulation.But trace equivalence relates behaviors that are sometimes not seen as being equivalent such as behaviors with deadlock and behaviors without deadlock. And quite remarkably bisimulation is rather easy to compute and trace equivalence on average is rather harder to compute by computers.

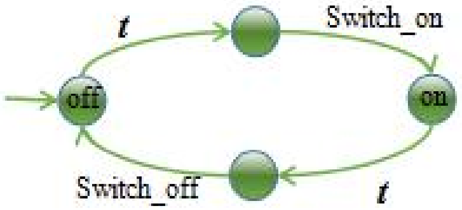
THE INTERNAL ACTION

Internal action is an action that takes place but cannot be observed directly.

I. Example

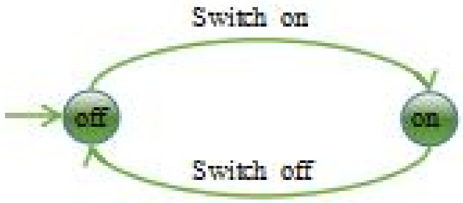


In this transition system you see that there's a double action of Switch_on and Switch_off. And if we trace the action it would be like this:
Trace: {(off), (off, Switch_on), (off, Switch_on, Switch_on), (off, Switch_on, Switch_on, on), ….)}
but using an internal action it would become like this:



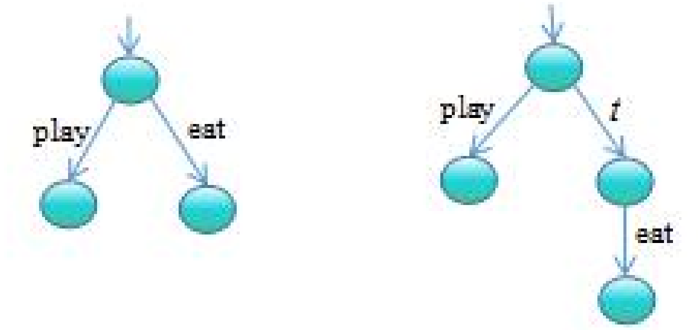
Using a notion invented by Robin Milner. Namely the hidden action, often written as a tau. The hidden or internal action is an action that can happen, but which we cannot directly observe, It happens silently without us seeing it. And in this way, we can hide the actions, and change the behavior. In this case what will happen is that, switch off cannot be seen, and then becomes tau action. And switching the light on cannot be observed, and then becomes a tau action as well. Just like the example above.

And to make it simpler we can also make the transition system like this:



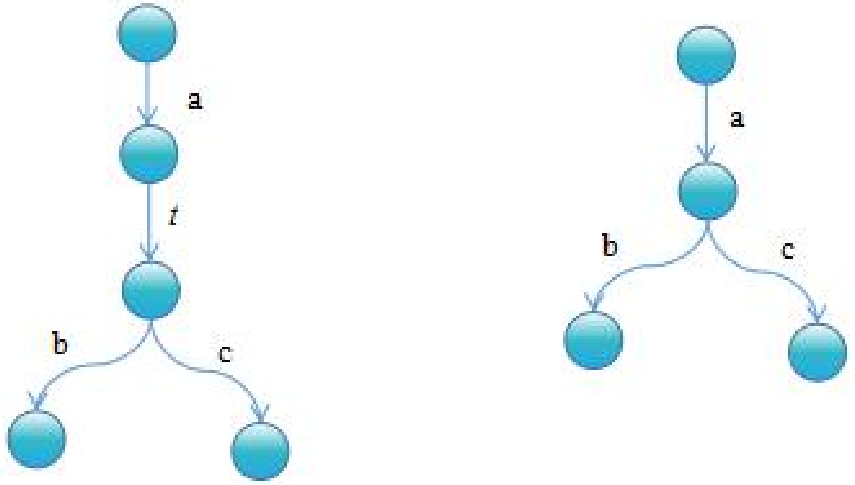
What we did is we removed the two taus because nobody is able to observe that they are there .and it become simpler, And simpler behavior's always convenient if you want to study it.

II. Another example:



In this example you can actually use a tau action, on the first transition you can do directly an action either you play or eat, but on the second transition system we have a tau, and lets say that the tau action is a rest action, either you can directly play or you can just rest (tau) first before eating, you can also stay on the tau if you wanted to.

III. Example



Internal actions are never observable. And we can always remove them and simplify the behavior. Just like the example above.

In many cases, we can remove taus but it's not always the case. But the fact that we can remove actions is really an important tool to that insight in complex behavior of systems.

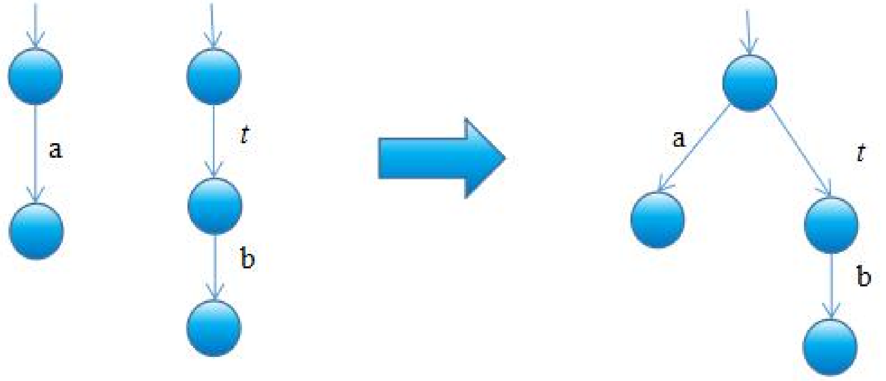
ROOTED BRANCHING BISIMULATION

OBJECTIVES:

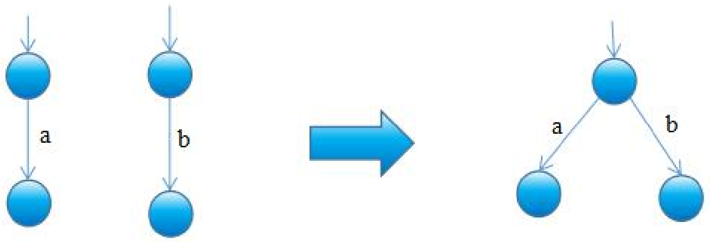
- To understand the concept of rooted branching bisimulation
- To identify whether a rooted branching bisimulation is similar to a branching bisimulation
- To understand the notion of root branching bisimulation to resolve a problem in a behavior.

we can actually combine a behaviours but sometimes, some problem occur with branching bisimulation.

Example:
if you're combining a two behaviours it would be like this:

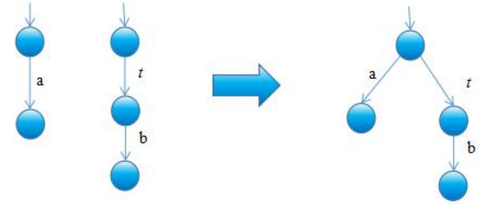


We can also simplify it by removing the tau action:

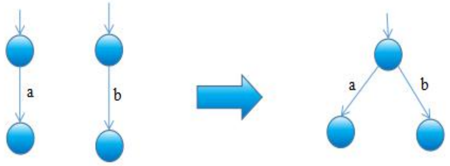


On the first look we can say that its normal and simple combination of behaviours but if we apply rooted branching bisimulation, we can actually see that there's a difference.

I.



II.

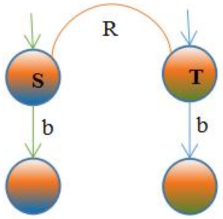


TRACES:
I. Trace: {{}, a}, {{}, b}, {{}, a, b}
II. Trace: {{}, a}, {{}, b}, {{}, a, b}

As you can see both has the same traces but if we compare directly the combined transition system, we can say that they are not branching bisimilar.
It is a situation where we have two behaviors that are equal and if we combine them, we get something that is not equal.

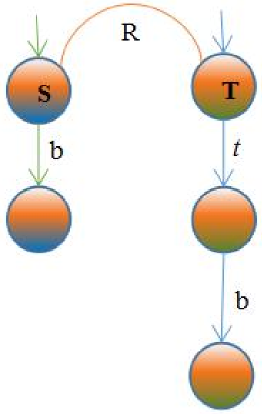
To further understand the concept of a rooted branching bisimulation here are some examples:

I. Example:



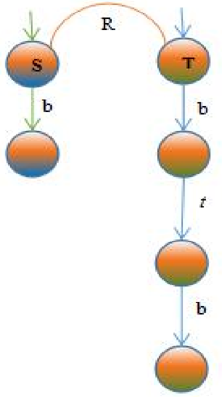
Are this two behaviour are rooted branching bisimilar? The answer is yes, because they have equal behaviours

II. Example:



Are this two behaviour are rooted branching bisimilar? The answer is no, they are not rooted branching bisimilar, we relate the two nitial states but if we mimic the action of the first branching bisimulation on the second bisimulation, you can see the difference wherein,on the first behaviour you will have an action doing the step b, but on the other behaviour, you will first have a tau action before doing the b step, this is a typical example of two transition systems that are branching bisimilar, but not rooted branching bisimilar.

III. Example:



Are this two behaviour are rooted branching bisimilar? The answer is yes, because as you can see we relate the two initial states, on the first initial state it has an action doing the step b, on the other hand, after the initial state there's also an action doing the step b, and after that another action has been done until it reached the final state. This two transition system, are rooted branching bisimilar because after the initial states both has an action doing the step b.

Remember: If two states are strongly bisimilar, then they are also rooted branching bisimilar, and if two states are rooted branching bisimilar, they are also branching bisimilar.

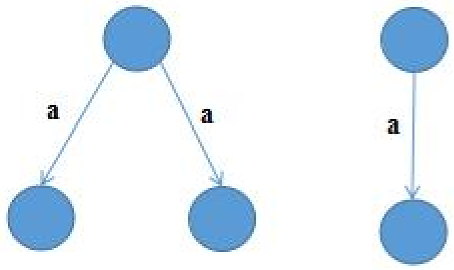
In conclusion branching bisimulation is not always preserved when we combine behaviors, that why we introduce the notion of root branching bisimulation as a solution of this problem.

ACTIVITY

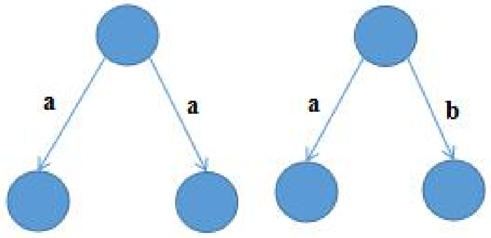
I. IDENTIFICATION

Instruction: Identify if the traces of following transition systems are equivalent.

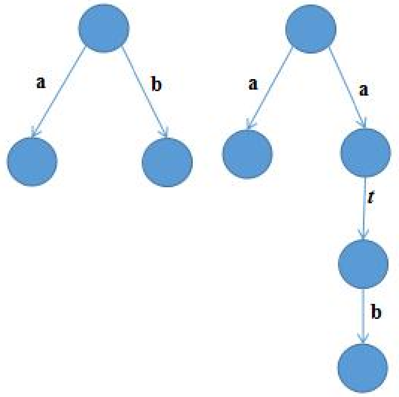
A. □ Yes □ No



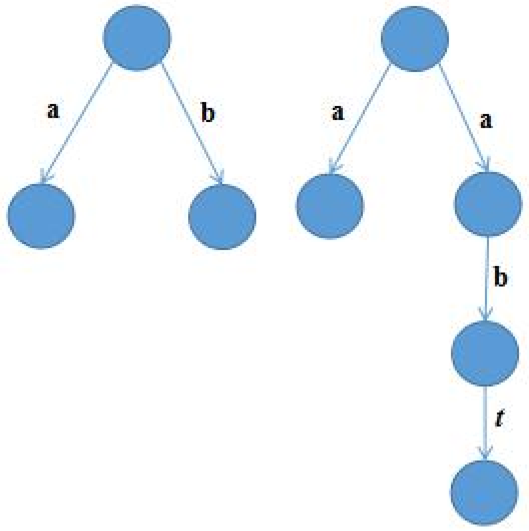
B. □ Yes □ No



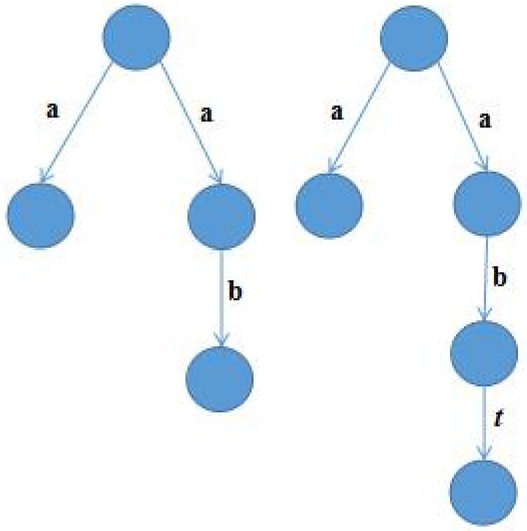
C. □ Yes □ No



D.□ Yes □ No



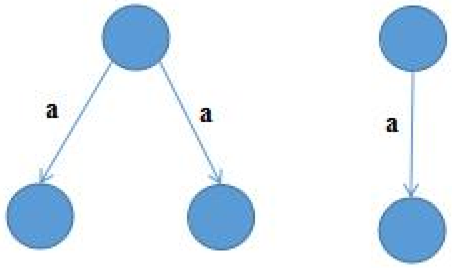
F.□ Yes □ No



II.
Instruction: Based on your answer on the first activity, list down the traces of each transition system and explain why can you say that the transitions systems are equivalent/ not equivalent.

A. Traces = ______________________________

Traces = ______________________________



Explanation:

____

____

____

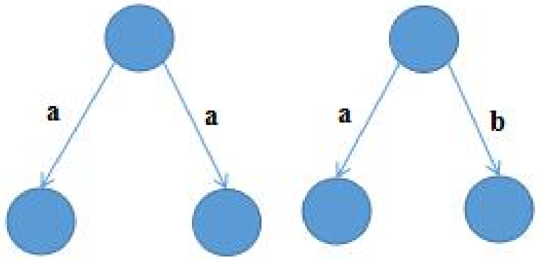
____

____

____

B. Traces = ______________________________

Traces = ______________________________



Explanation:

____

____

____

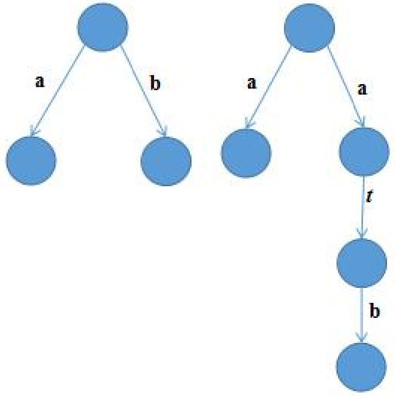
____

____

____

C. Traces = ______________________________

Traces = ______________________________



Explanation:

____

____

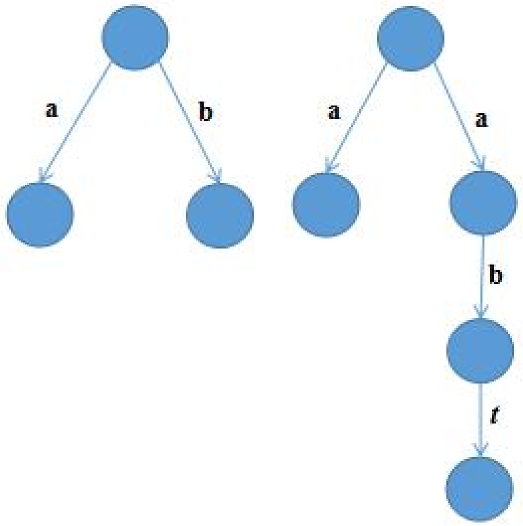
____

____

____

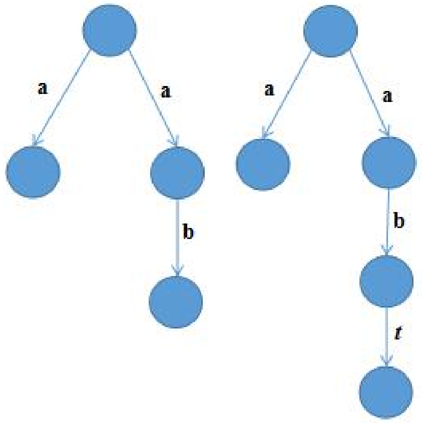
____

D. Traces = ______________________________________
Traces = ______________________________________



Explanation:

E. Traces = ______________________________________
Traces = ______________________________________

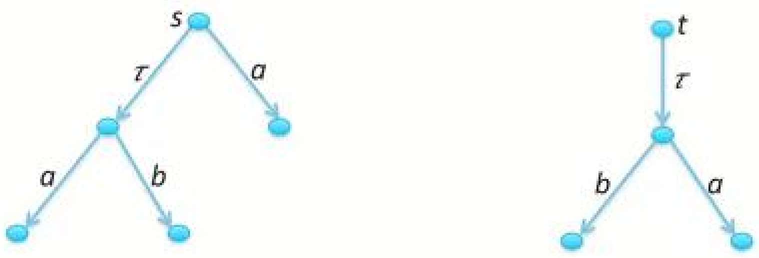


Explanation:

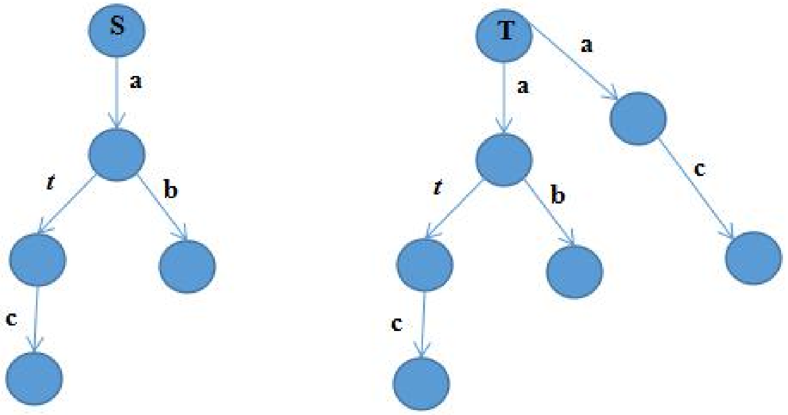
III.

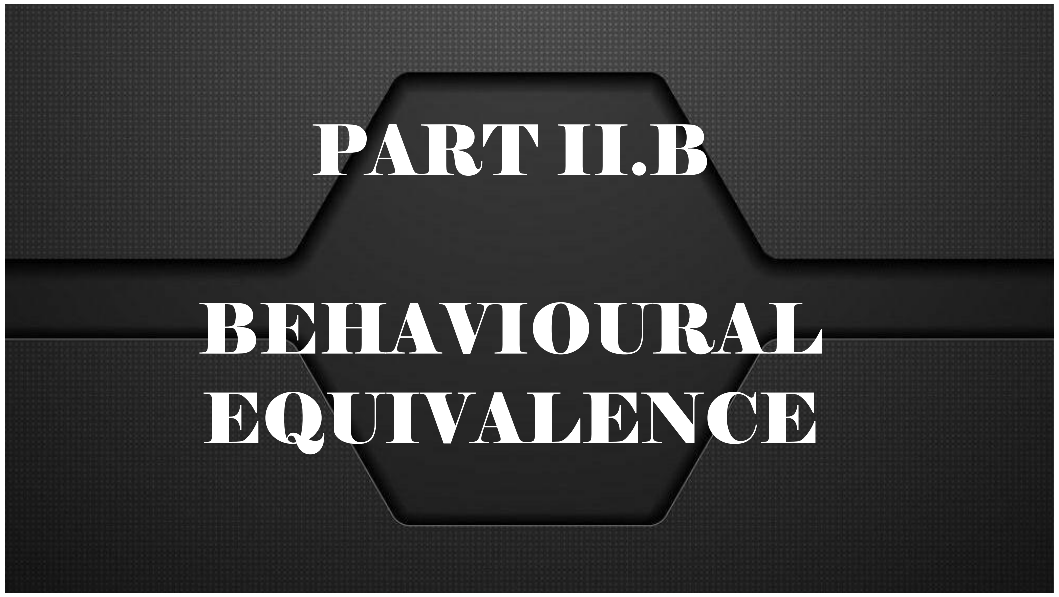
Instruction Identify if following transition systems are rooted branching bisimilar.

A. □ Yes □ No



B. □ Yes □ No



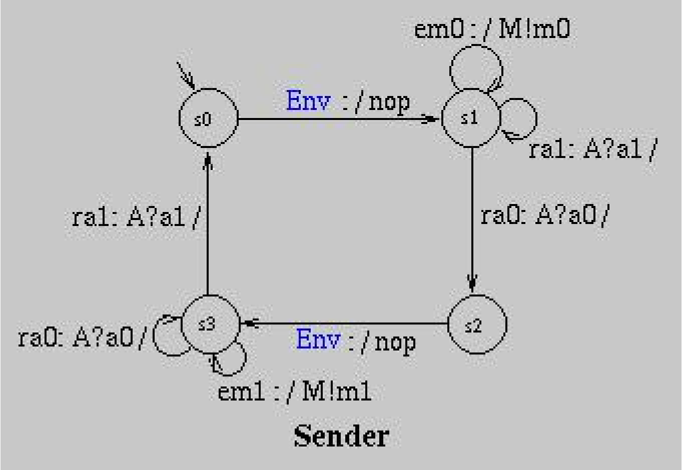

PART II.B

BEHAVIOURAL EQUIVALENCE

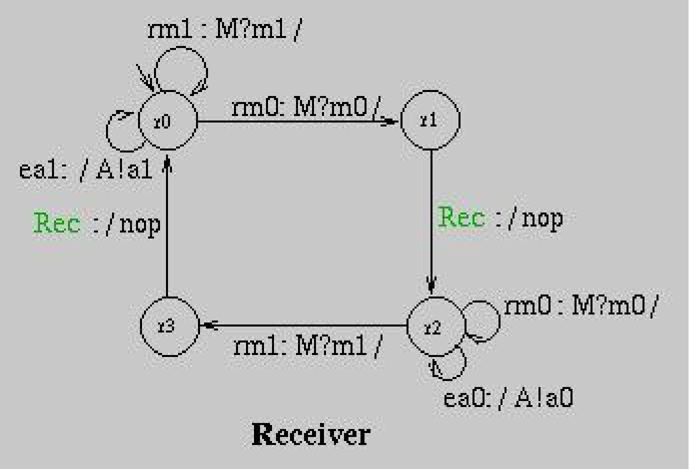
THE ALTERNATING BIT PROTOCOL

The alternating Bit Protocol is a reliable transport protocol of stop-and-wait family. It uses positive acknowledgement (ACK)and additional information of one bit to each message exchanged. The additional bit is used to identify a message in the sequence of messages sent. Only one bit suffices because no new data message is sent before the acknowledgement for the previous data message sent is not arrived (stop-and-wait).

The model we consider for this protocol is build from two automata (for sender and receiver) which exchange messages through two fifo lossy channels M (for messages sent from sender to receiver) and A (for acknowledgements sent from receiver to sender). The sender reads each message of the producer and sends it through M to the receiver with its additional bit. Then, it waits for an acknowledgement of this message (with the same bit) through A. If the acknowledgement does not arrive, the sender resends the message (with the same bit). When it receives the acknowledgement, the sender sends the next message read from the producer with an incremented bit



.



Protocol Parameters

The following settings are adequate for a simple simulation. For a more advanced exploration, choose different options and click Change Settings. This will cause the simulation to restart.

Optionally set the level of control that the simulator user needs over the medium:

Automatic:

Messages are delivered immediately without loss. This is suitable for initial experimentation, but limits what can be explored. For example, message are never lost and never have to be timed out and resent.

Delivery:

Alternatively, message delivery may be controlled - though messages will still never be lost. Choose this option to allow messages to be delivered in different orders and to be resent.

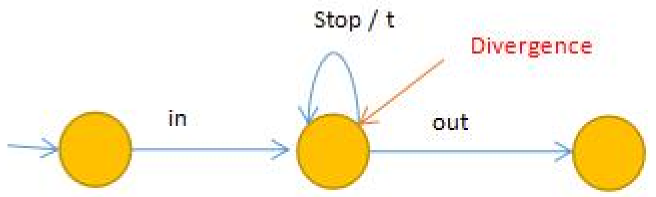
Delivery/Loss:

Finally, delivery and loss of messages may be completely controlled. This is the most comprehensive option, but also the most complex one to manage.

Protocol Simulation

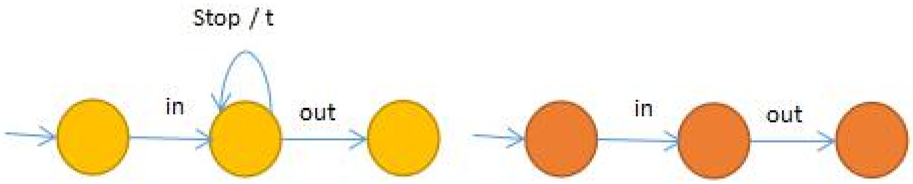
The protocol simulation shows a time-sequence diagram with transmitting and receiving protocol entities, and a communications medium that carries messages. The transmitter simply sends messages numbered DATA(0) or DATA(1); the content of messages is not identified. These are acknowledged with ACK(1) or ACK(0) respectively. Note that if a DATA message is received again due to re-transmission, it is acknowledged but discarded.

DIVERGENCE PRESERVING BRANCHING BISIMULATION



We have here an example of transition system that has 3 states with an action in and out and an infinite loop of stop that represents a tau.

So it behaves like this, with an in and out and you can do an arbitrary number of stops in-between. The possibility to do an infinite sequence of taus is called a divergence.



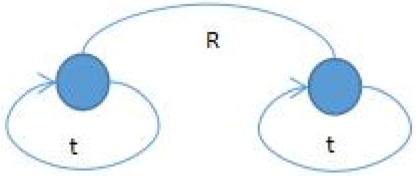
Now we have two transition systems and this two transition systems are the same, we just removed the loop of stop or the tau, because Branching bisimulation removes divergencies.

A property of branching bisimulation is the following:

- That all states that you have on a tile loop will be merged
- and the tau loop will be removed.

To improve the situation we would like to have a notion of branch bisimulisation that recognizes divergencies explicitly. And therefore we define the notion of divergence preserved through branching bisimulation. And that is just ordinary branching bisimulation with one extra property

A. Example:



Based from the example, we have two states that are related to each other therefore if one state can do an infinite number of taus, the other statements should also be able to do an infinite sequence of taus. And it's the only extra condition that we add to divergence preserving branching simulation.

Summary:

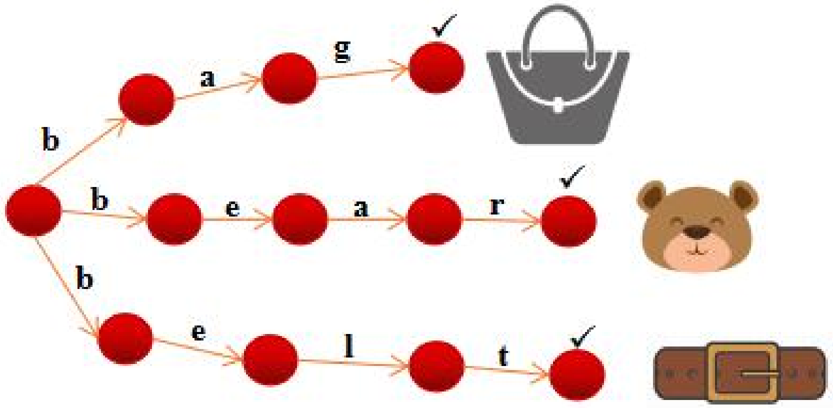
- Divergences as infinite sequences of internal steps
- Branching bisimulation does not preserve divergencies
- If divergent behaviour is important, divergence preserving branching bisimulation must be used

LANGUAGE FAILURE AND COMPLETED TRACE EQUIVALENCE

Label transition systems are traditionally used not in the study of behavior of processes, but in the study of languages and compilers, parsing, and scanning.

Suppose we have a text and we want to recognize the word bag, we also want to recognize the word belt and the word bear.

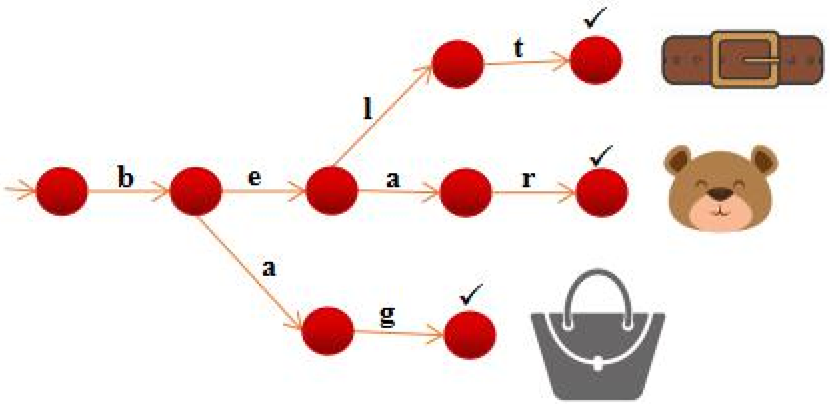
If we convert it on a label transition system it would be like this:



Here we have words and that are typically traces ending in a termination symbol. Two transition systems are equal if they recognize exactly the same words, namely, beside the two transition systems are language equivalent if they have exactly the same set of words.

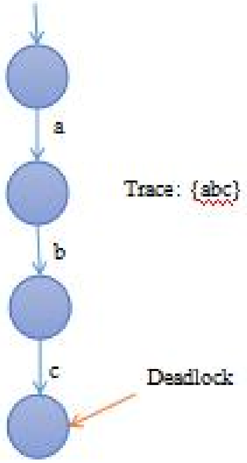
But as you observe the transition system carefully you'll notice that a problem can occur in this kind of transition system. Where in if you expect the word bear then you take a branch of this transition system. But if the second letter then turns out to be an A, we have to backtrack and take the second branch, and that is inefficient.

What would be efficient is to make a deterministic transition system that is exactly language equivalent to this original. And that can be done in this way:



this is an efficient way to recognize the word immediately. The question to make scanning efficient is that you simply have to constrict a language equivalent deterministic labeled transition system.Which is equivalent to the set transition system recognizing the set of words.

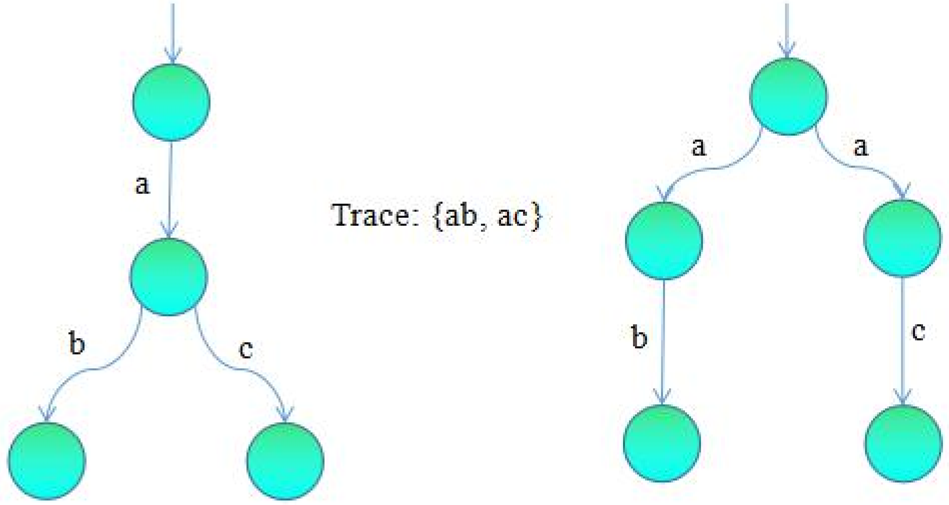
Another Example:



A completely different equivalence is completed trace equivalence, Suppose we have a behaviour where there's an a, moving on a state where there's an action b, moving forward on a state where there's an action c, and because we do not observe anymore then we can say that the system is in deadlock.

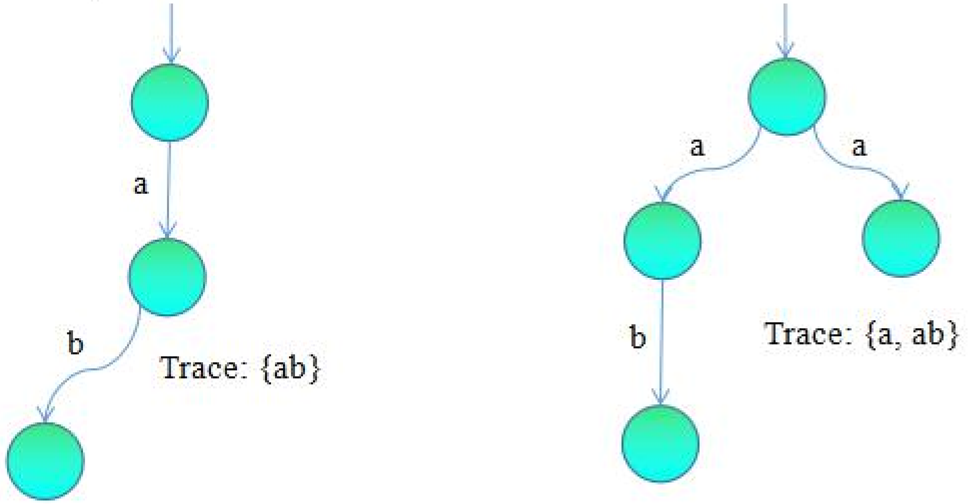
What we can say is that the completed trace of the system is typically every trace that ends in a deadlock or that ends in a terminating state.

I. Example:



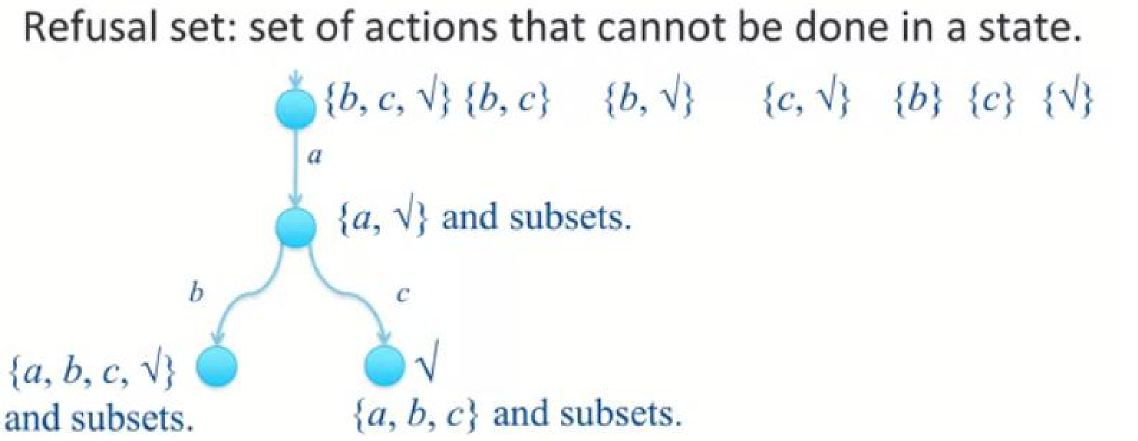
This is an example of a completed trace equivalence because the traces of these two behaviour are {ab, ac}.

II. Example



Here is then an example of not completed trace equivalence, actually it is the same on the first example, the only thing is that we removed the c action, and because of that the transition become a not complete trace equivalence, because on the first transition only an {ab} action can be perform, how ever on right transition it can do a two action {a, ab}, and based on the traces of these two transition it clearly states that they're not equivalent.

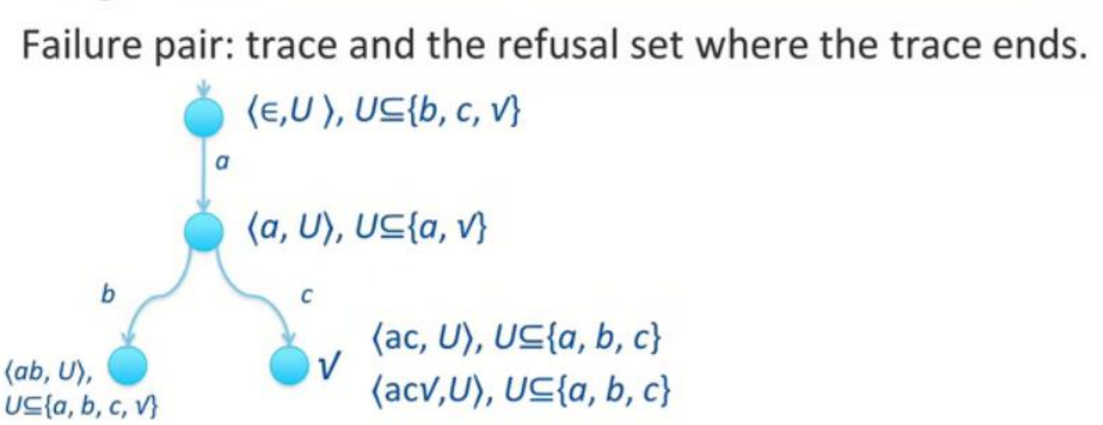
Now we proceed to another equivalence which is called as Failure(s) Equivalence, It is the complete trace congruence under standard operators that are applied on behavior, It starts with the so-called notion of a refusal set.


Refusal set: set of actions that cannot be done in a state.

{b, c, √} {b, c} {b, √} {c, √} {b} {c} {√}
a
{a, √} and subsets.
b
c
{a, b, c, √} and subsets.
√
{a, b, c} and subsets.

In initial state of the diagram, we cannot do the b, the c, and we can also not terminate. But there's another refusal set because we can also not do a b and a c. And actually any subset of this largest refusal set is also a refusal set. Even the empty set is a refusal set. In the second state, we can see that we cannot do an a and a termination. And that means that a and terminations are all refusal sets. And in this state at lower left part, we see that {a, b, c, tick} are actions that cannot be done. And here we can see that {a, b, c} are actions that cannot be done. So this set and all their subsets, these two sets and their subsets are refusal sets

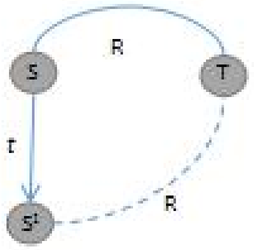
Given that we know what refusal sets are, we define a notion of a failure pair. And a failure pair is the following.

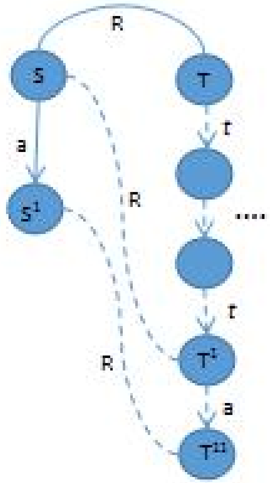

Failure pair: trace and the refusal set where the trace ends.

WEAK BISIMULATION

In 1918 Robin Milner defined a weak bisimulation, As an equivalent as a bisimulation for a hidden actions. And in 1989, Glabbeek and Weijland defined a notion of branching bisimulation. And actually, weak bisimulation and branching bisimulation are quite similar.

In branching bisimulation, even if you have more than one state and only has a tau action then its can be possible if you relate it with one state for example:

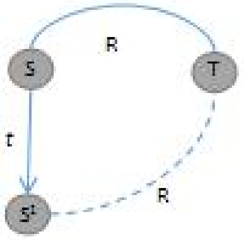




Another case in branching bisumulation was that if we could do a step from two related states. And could do an A step, and mimic that, then it could be mimicked by a number of taus and then an A step. The final states are related and the state before the action a should also be related on the S state. Just like the transition system on the right

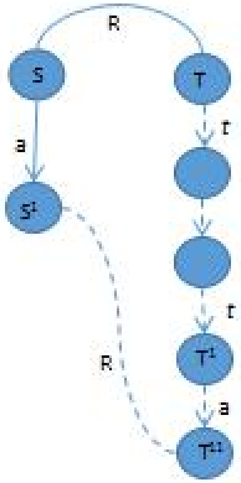
.

In weak bisimulation, the first case is exactly the same so if we do this tau, we can mimic it by doing nothing and add points that should be related.



And then here there's a difference if we have two states and these are related. And we do an A step in one state. It's allowed to mimic it by doing a number of taus. Then doing an A. Then doing again a number of taus. And then only the endpoints should be related. So, and this is very different than branching bisimulation.

Example:



In addition compared to branching bisimulation, Branching bisimulation can be calculated far more effectively than weak bisimulation.

WEAK TRACE EQUIVALENCE

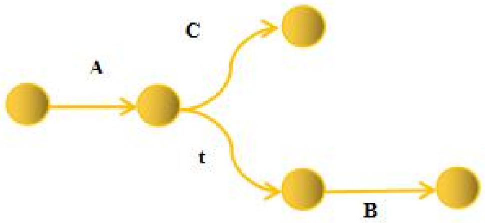
Objectives:

- To identify what is weak bisimulation
- To understand the essentials of understanding weak bisimulation

Weak Trace bisimulation ignores silent transitions in a very general way. It requests that a transition labeled with an action is simulated by a transition labeled with the same action but preceded and followed by a sequence of τ transitions.

Two transition systems are weak trace equivalent if, and only if they have the same set of weak traces. And weak traces are those traces that tiles are omitted.

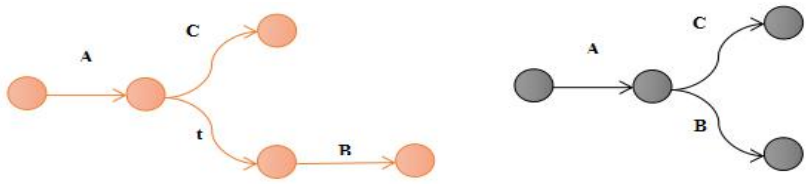
A. Example:



If you look at this example. We have the following weak traces. Namely the empty traces weak trace. The tracer to A. AB but not A tao B we omit the tao. And AC are the tracers of this system.

Note: You should only apply weak trace reductions on relatively small transition systems.

B. Example



Based on these two transition system. you are asked the answer the question, are these two transitions systems weak trace equivalent?

☐ Yes ☐ No

It's obvious that they are equivalent, because they have the same set of weak traces. And you can immediately see that weak trace equivalence has a number of unpleasant properties. Look for instance at the b at the left. You can see that you can end up in a state where you can do the b, but not an a. But this is not what's happening at the right site. The branching structure of processes is not preserved by the trace equivalent.

That is why we are always a little bit hesitant in using it. But sometimes it is useful, because if you have a small transition system applying weak trace equivalence can make it even smaller and, therefore, more insightful.

Advantages:

- Weak trace reduction allows to remove all the internal action from the behavior

Disadvantages:

- There is no unique minimal transition system
- If you apply a weak trace reduction. Then it can be very hard to calculate, and it can be very time consuming.
- And it can actually result in much bigger transition systems.

When to use which Behavioural Equivalence?

Objectives:

- To know when we have to use which equivalence.
- To identify the major process equivalences.
- To know when should we apply behavioral reduction

Different behavioral equivalences:

Strong bisimulation, is one of the very convenient bisimulation to use, because it is always safe. And if you do not know what to use, restrict to strong bisimulation.

Trace equivalence, on the other hand, is attractive because it can reduce the behaviour much more. But it will not preserve the branching behavior, for instance, deadlocks.

Branching behaviour, is were we consider two system as equivalent only if every computation, that is, every alternating sequence of (visible and silent) actions and states, of one system has a correspondent in the other. It is where we have seen that we would not like to see these as equivalent. But from the perspective of trace equivalents, they are equal.

Failures equivalence preserves the deadlocks. It preserves of some the branching structure of processes, and in particular, it preserves equality under the application of ordinary process operators.

Also **Divergence preserving branch bisimulation** is always safe to use. It removes all internal steps, but it preserves all the other structure of the behavior. If you're not interested in divergences, so we find it's allowed to remove tau-loops. Then we can actually also apply branching bisimulation. And tau-loops can typically can mean that there is some internal activity going on that takes a lot of time.

Weak bisimulation is for all practical purposes equal to branching bisimulation. Whether you use branching bisimulation or weak bisimulation in the settings up till now, it really doesn't matter.

Weak trace equivalence on the other hand can be very attractive because it removes all the taus from the behavior. And f you have many taus, but would like to get rid of them, this is the equivalence to use. But it does not preserve the branching structure of the behavior, and that means that the behavior can really look different after applying this.

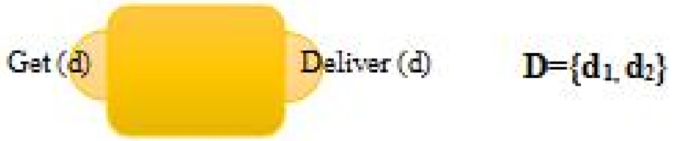
TRANSITION SYSTEM WITH DATA, TIME AND PROBABILITIES

Objectives:

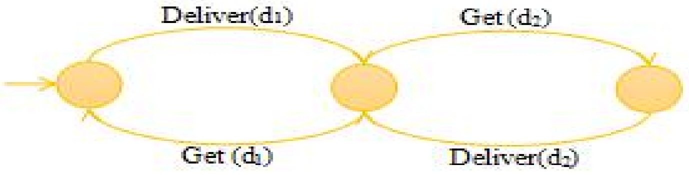
- to indentify different extensions of labeled transition system.
- to know the purpose of a   Label Transition Systems

Label Transition Systems are used for many more purposes and therefore there are many extensions of label transition systems. There are different extensions to transition systems and one of the extension is extension with data.

I. Example:

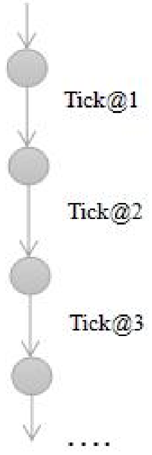


The idea is that with every action we can simply take a datum after it, between breakage, so here we see, get(d). That means we get a particular datum of a particular sort. And if the data is finite, and the data consist of datum one and datum two.



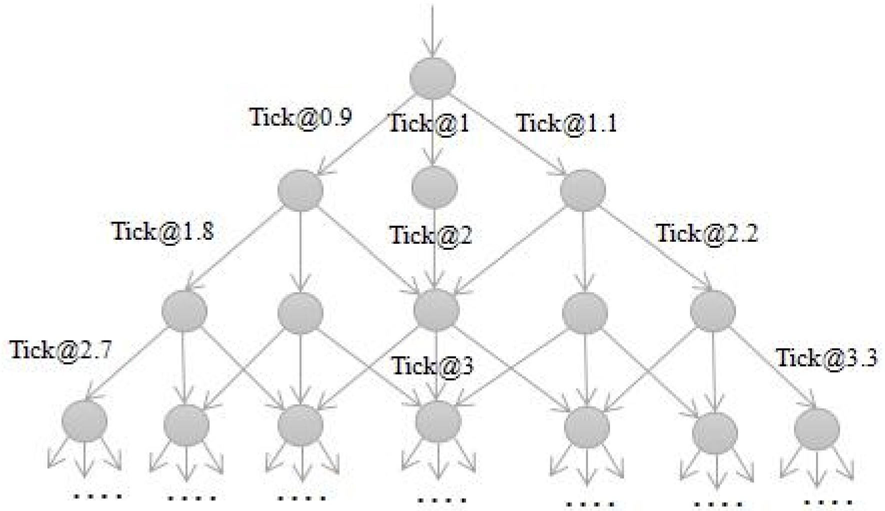
Here is a transition system and typically a potential transition system that could describe the behavior of this system. It is where we actually read data one and then deliver data one. Or we can decide to read, get data two and deliver data two. And in both cases after delivery go to the initial state. Get a next data and deliver it again. Most important point is that you can add data to actions in this way and so describe components that sent data back and forth.

An alternative extension is the extension with time. So there are many different forms in which time is added to transition systems. A very straightforward one is this where you typically add a time-stamp to an action.

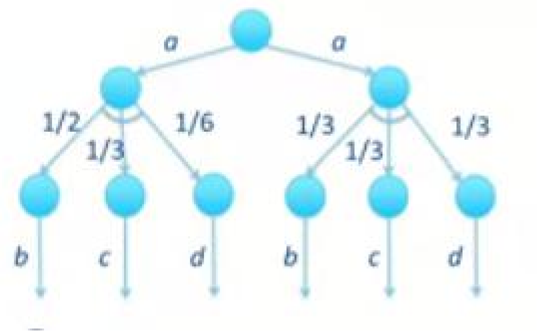


And we can describe this clock. And this clock ticks, and it ticks at time one, and then at time two, and then at time three and this goes on indefinitely. So this is an infinite transition system.

One of the properties of describing slightly more realistic behavior is that it quickly can become complex. Assuming that we have this clock but it's not completely a stable clock. So it can take a time, one, two, three, etc. But it can also click a tick a little bit earlier. Namely, X times 0.9 or it can take a little bit too late X time 1.1 and this can go on forever. So the typical behavior then becomes this. It can have two early ticks and then the second tick is at time 1.8 or it can have two late ticks, or it can have any combinations of late and early ticks. And then, this is typically the behavior that you get. So here it grows and it grows even more. And this makes us feel quite intriguing, because the standing such complex behavior is often a quite difficult task.



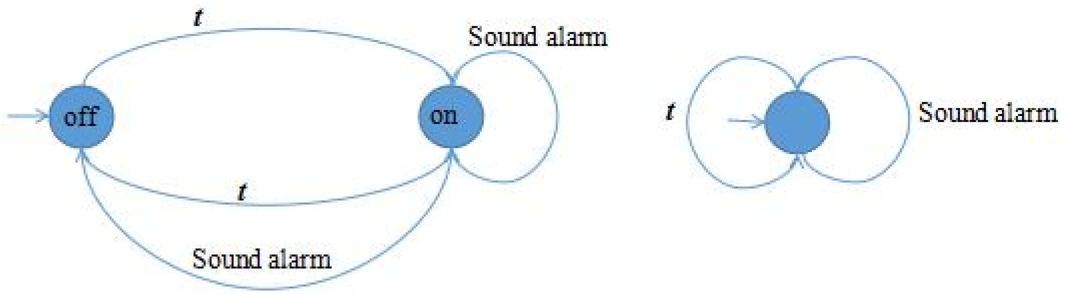
Another extension is an extension with probabilities. Currently this is heavily under investigation in the scientific community and the idea is the following. We can have actions in the same way as we had to before but we can now also have probabilistic choices and that's often denoted like this. So, this means that you go after doing the A, the probability a half to which state where you can do a B.



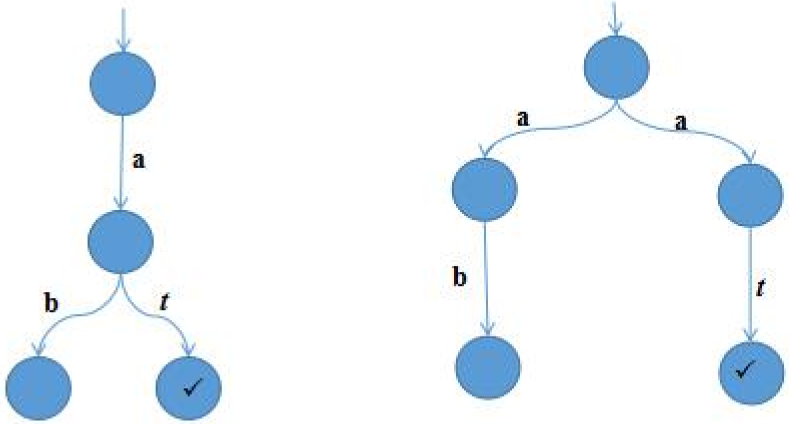
With one-thirds to a state where you can do a C. And with one-sixth to a state where you can do a D. And this second choice, is really a probabilistic choice in the sense of probability theory. So we have a very strange situation where we have normally terministic choices that we cannot influence and we have probabilistic choices that are governed by probabilities. So this is very weird behavior, where you can normally terministically go to a state with one distribution or to a state with a different distribution.

ACTIVITY

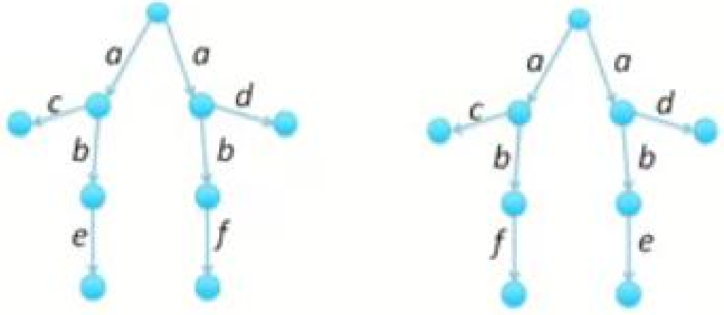
1. Is the behavior of this two transition system are equal, in the sense of divergence sensitive branching bisimilarity?



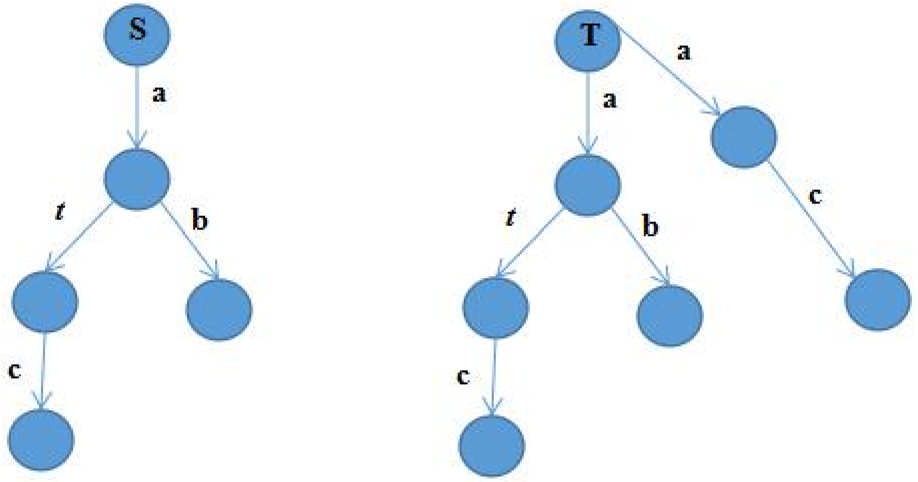
2. Are these two transition systems language equivalent?



3. Are these two transition systems are they completed trace equivalent?



4. Are these two transition systems weak bisimilar?



I. If you fully understand all the lessons create the DFA and an NDFA of the following traces:

1. Trace: {{}, a, ab, abc, abcd, aab, aabbc, aabbcc, aabbccdd….}
2. Trace: {{}, 1, 10, 100, 1000, 10000, …}
3. Trace: {{}, 1, 10, 101, 110, 1101, 1110, 11101, ….}