Chapter 2

Actions, behaviour, equivalence and abstraction

In this chapter the basic notions of (inter)action and behaviour are explained in terms of transition systems. It is discussed when different transition systems can behave the same and it is indicated how complex behaviour can be abstracted by hiding actions.

2.1 Actions

Interaction is everywhere. Computer systems, humans, machines, animals, plants, molecules, planets and stars are all interacting with their environment. The classical view is that such interactions are continuous such as gravity pulling stellar objects towards each other. There is a long mathematical tradition in researching such continuous communication.

But many systems communicate by having infrequent, short interactions. We use the term discrete interactions to contrast such communications with the classical continuous interaction. Human communication is a typical example. People meet, shake hands, exchange messages and proceed to communicate with others. The communication in and especially among computers has the same pattern. The whole nature of computers seems to be to receive, process and send messages. And as many of the systems that we make these days contain computers inside, they can be viewed as discretely interacting systems.

The complexity of message exchanges among computerised systems is steadily increasing and has reached a level where it is very hard to understand. This complexity needs to be tamed by making models and having the mathematical means to understand these models. The first purpose of this book is to provide the modelling means and mathematical analysis techniques to understand interacting systems. The second derived purpose is to provide the means to design such systems such that we know for sure that these systems work in the way we want.

Basic actions, also called interactions, represent the elementary communications and are the main ingredients of such models. We denote them abstractly by letters a,

b, c or more descriptively by names such as $read$, $deliver$, and $timeout$. They are generally referred to as *actions* and they represent some observable atomic event. For instance, action $deliver$ can represent the event of a letter being dropped in a mailbox. Action *read* can consist of reading a message on a computer screen.

Actions can be parameterised with data; an action a taking data parameter d is denoted by $a(d)$; examples of such actions are $read(1)$, $write(true, 2)$ and $draw(1.5, 2.5, sqrt(2))$, where $sqrt(2)$ is a data expression denoting the square root of 2. This feature is essential in modelling reactive systems that communicate data and make decisions based on the values of communicated data.

The fact that an action is atomic means that actions do not have distinct start and end moments and hence, cannot overlap each other. For every pair of actions a and b, the one happens before the other, or vice versa. Only in rare cases a and b can happen exactly at the same moment. We write this as $a|b$ and call this a *multi-action*. It is possible to indicate that an action can take place at a specific time. E.g., $a@3$ denotes that action a must take place at time 3. For the moment multi-actions and time are ignored. We come back to it in chapters 4 and 8, respectively.

We use an alarm clock as our running example throughout this chapter. Our simple alarm clock has three basic actions, namely, *set*, *reset* and *alarm*. Also, we specify a variant of alarm clock in which the number of alarms can be given to the clock; in this variant we use actions of the form $set(n)$, where n is a natural number denoting the number of times the alarm should go off.

Exercise 2.1.1. What are the actions of a CD-player? What are the actions of a text-editor? And what of a data-transfer channel?

2.2 Labelled transition systems

The order in which actions can take place is called behaviour. Behaviour is generally depicted as a labelled transition system, which consists of a set of states and a set of transitions labelled with actions that connect the states. Labelled transition systems have an initial state, which is depicted by a small incoming arrow. They may have terminating states, generally indicated with a tick (✓). In figure 2.1, the behaviours of two simple processes are depicted. Both can perform the actions a, b, c and d. At the end, the lower one can terminate, whereas the upper one cannot do anything anymore; it is said to be in a *deadlock*, i.e., in a reachable state that does not terminate and has no outgoing transitions.

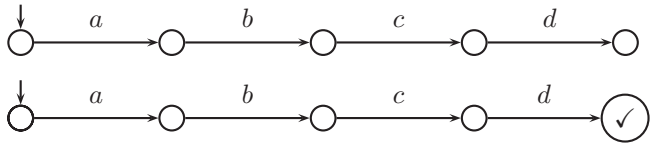


Figure 2.1: Two simple linear behaviours of which the lower one can terminate

Such simple diagrams are already useful to illustrate different behaviours. In figure

2.2 the behaviours of two alarm clocks are drawn. The behaviour at the left allows for repeated alarms, whereas the behaviour at the right only signals the alarm once. Note also that the behaviour at the left only allows a strict alternation between the set and the $reset$ actions, whereas this is not the case in the right diagram as for instance the trace $set\ alarm\ set$ is possible.

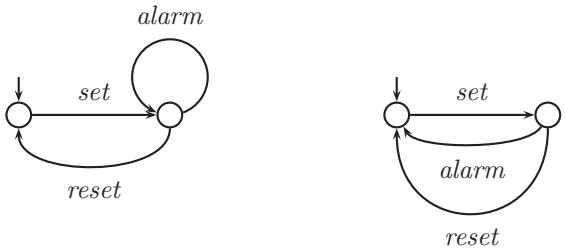


Figure 2.2: Two possible behaviours of an alarm clock

Our model of the alarm clock can be extended with a feature for specifying the number of alarms to go off. This is achieved by parameterising the action set, with a natural number as depicted in figure 2.3. In this picture we assume that the pattern of behaviour is repeated and hence, we have an infinite labelled transition system.

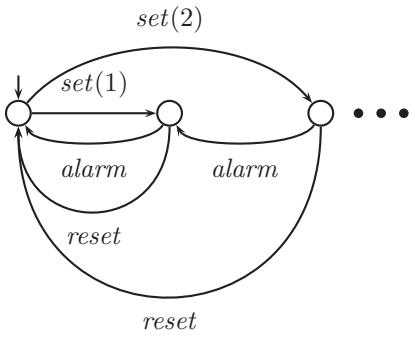


Figure 2.3: An alarm clock with a multiple alarm feature

A state can have more than one outgoing transition with the same label to different states. The state is then called *nondeterministic*. A deterministic transition system contains no reachable nondeterministic states. Nondeterminism is a very strong modelling aid. It allows to model behaviour of which the details are not completely known, or are too complex to be modelled fully. The first use of nondeterminism is called underspecification. The second use is called abstraction allowing for simple models of complex phenomena.

In figure 2.4 sounding the alarm is modelled using nondeterminism. If an alarm sounds, you cannot tell whether it is the last one, or whether there are more to follow. This can be intentionally underspecified, allowing an implementer to build an alarm clock which can sound the alarm a fixed (yet arbitrary) number of times. But figure 2.4 can also represent that the alarm sounds exactly 714 times before stopping. When the

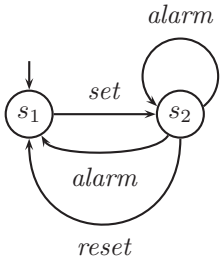


Figure 2.4: Nondeterministic behaviour of an alarm clock

fact that the alarm sounds exactly 714 times does not outweigh the increased complexity of the model, this abstract nondeterministic transition system is a good model.

Robin Milner[1] was one of the early defenders of the use of nondeterminism [131, 133]. He called it the weather condition. The weather determines the temperature. The temperature influences the speed of processors and clocks in a computer. This may mean that a timeout may come just too late, or just too early for some behaviour to happen. It generally is not effective to include a weather model to predict which behaviour will happen. It is much more convenient to describe behaviour in a nondeterministic way.

The formal definition of a labelled transition system is the following.

Definition 2.2.1 (Labelled Transition System). A *labelled transition system* (*LTS*) is a five tuple $A = (S, Act, \longrightarrow, s, T)$ where

- S is a set of *states*.
- Act is a set of actions, possibly multi-actions.
- $\longrightarrow \subseteq S \times Act \times S$ is a *transition relation*.
- $s \in S$ is the *initial state*.
- $T \subseteq S$ is the set of *terminating states*.

It is common to write $t \xrightarrow{a} t'$ for $(t, a, t') \in \longrightarrow$.

Often, when not relevant or clear from the context, the set T of terminating states and/or the initial state are omitted from the definition of an *LTS*.

Exercise 2.2.2. Make the following extensions to the alarm clock.

1. Draw the behaviour of an alarm clock where it is always possible to do a *set* or a *reset* action.
2. Draw the behaviour of an alarm clock with unreliable buttons. When pressing the *set* button the alarm clock can be set, but this does not need to be the case. Similarly for the *reset* button. Pressing it can reset the alarm clock, but the clock can also stay in a state where an alarm is still possible.

[1] Robin Milner (1934-2010) was the developer of the theorem prover LCF and the programming language ML. He is also the founding father of process calculi. In 1991 he received the Turing Award.

3. Draw the behaviour of an alarm clock where the alarm sounds at most three times when no other action interferes.

Exercise 2.2.3. Describe the transition system in figure 2.4 in the form of a labelled transition system conforming to definition 2.2.1.

2.3 Equivalence of behaviours

When do two systems have the same behaviour? Or stated differently, when are two labelled transition systems behaviourally equivalent? The initial answer to this question is simple. Whenever the difference in behaviour cannot be observed, we say that the behaviour is the same. The obvious next question is how behaviour is observed. The answer to this latter question is that there are many ways to observe behaviour and consequently many different behavioural equivalences exist. We only present the most important ones here.

2.3.1 Trace equivalence

One of the coarsest (most unifying) notions of behavioural equivalence is *trace equivalence*. The essential idea is that two transition systems are equivalent if the same sequences of actions can be performed from their respective initial states. This corresponds to observing that actions can happen without interacting with them. It is not even possible to tell whether a system is deadlocked or whether more actions will come.

Traces are sequences of actions, typically denoted as $a_1a_2a_3 \ldots a_n$. We typically use letters σ and ρ to represent traces. The termination symbol $\checkmark$ can also be part of a trace (usually appearing at its end). The symbol ϵ represents the empty trace.

Definition 2.3.1 (Trace equivalence). Let $A = (S, Act, \longrightarrow, s, T)$ be a labelled transition system. The set of *traces* (*runs*, *sequences*) $Traces(t)$ for a state $t \in S$ is the minimal set satisfying:

1. $\epsilon \in Traces(t)$, i.e., the empty trace is a member of $Traces(t)$,
2. $\checkmark \in Traces(t)$ iff $t \in T$, and
3. if there is a state $t' \in S$ such that $t \xrightarrow{a} t'$ and $\sigma \in Traces(t')$ then $a\sigma \in Traces(t)$.

The set of traces of the labelled transition system is $Traces(s_0)$. Two states $t, u \in S$ are *trace equivalent* if and only if (iff) $Traces(t) = Traces(u)$. Two transition systems are *trace equivalent* iff their initial states are trace equivalent.

The sets of traces of the two transition systems in figure 2.1 are respectively $\{\epsilon, a, ab, abc, abcd\}$ and $\{\epsilon, a, ab, abc, abcd, abcd\checkmark\}$. The two transition systems are not trace equivalent.

Consider the labelled transition systems for the two alarm clocks depicted in figure 2.5. These two systems are trace equivalent: the set of traces for both labelled transition

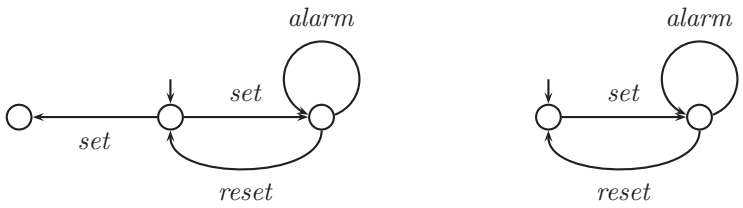


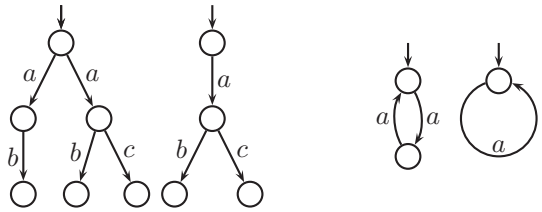
Figure 2.5: Two trace-equivalent alarm clocks

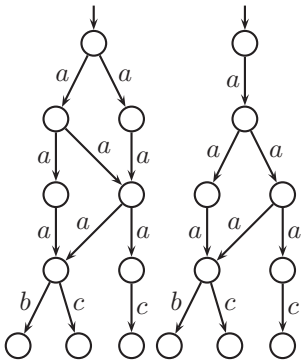
systems is: $\{set,\ set\ alarm^\star,\ (set\ alarm^\star\ reset)^\star\}$, where * is the so-called Kleene star and denotes zero- or more times repetition (zero-time repetition results in the empty trace).

The alarm clock at the left-hand side has a nondeterministic choice between the two transitions labelled with set: if it moves with the set transition to the left, it deadlocks. At the right there is no deadlock after the set action. As one can generally observe a deadlock, the observational behaviour of the two transition systems is different. This is the reason why trace equivalence is not used very often and finer notions of equivalence are used that take deadlocks into account.

However, there are cases where trace equivalence is useful, especially when studying properties that only regard the traces of processes. A property can for instance be that before every $reset$ an set action must be done. This property is preserved by trace equivalence. In order to determine this for the transition system at the left in figure 2.5, it is perfectly valid to first transform it into the transition system on the right of this figure, and then determine the property for this last transition system.

Exercise 2.3.2. Which of the following labelled transition systems are trace equivalent.





2.3.2 ★Language and completed trace equivalence

In language theory labelled transition systems are commonly used to help in parsing languages. Generally, the word automaton is used for labelled transition systems in that context. Process theory, as described here, and language theory have a lot in common. For instance, grammars to describe languages are essentially the same as process expressions, described in the chapter 4.

There is however one difference. In the process world there are many different behavioural equivalences, whereas in the language world *language equivalence* is essentially the only one. Every trace that ends in a successful state is a sentence. Such a sentence is said to be accepted by the language. Two processes are language equivalent if their sets of sentences are the same. More formally:

Definition 2.3.3 (Language equivalence). Let $A = (S, Act, \longrightarrow, s, T)$ be a labelled transition system. We define the language $Lang(t)$ of a state $t \in S$ as the minimal set satisfying:

- $\epsilon \in Lang(t)$ if $t \in T$, and
- if $t \xrightarrow{a} t'$ and $\sigma \in Lang(t')$ then $a\sigma \in Lang(t)$.

Two states $t, u \in S$ are *language equivalent* iff $Lang(t) = Lang(u)$. Two labelled transition systems are *language equivalent* iff their initial states are language equivalent.

The language of the first automaton in figure 2.1 is the empty set. That of the second transition system is $\{abcd\}$.

The closest pendant in the process world is *completed trace equivalence*. A completed trace is a sequence of actions that ends in a deadlocked or a terminating state (and the difference between these states can be observed).

Definition 2.3.4 (Completed trace equivalence). Let $A = (S, Act, \longrightarrow, s, T)$ be a labelled transition system. We define the completed traces $CompletedTraces(t)$ of a state $t \in S$ as the minimal set satisfying:

- $\epsilon \in CompletedTraces(t)$ if $t \notin T$ and there are no $t' \in S$ and $a \in Act$ such that $t \xrightarrow{a} t'$,

- $\checkmark \in CompletedTraces(t)$ if $t \in T$, and
- if $t \xrightarrow{a} t'$ and $\sigma \in CompletedTraces(t')$ then $a\sigma \in CompletedTraces(t)$.

Two states $t, u \in S$ are *completed trace equivalent* iff $Traces(t) = Traces(u)$ and $CompletedTraces(t) = CompletedTraces(u)$. Two labelled transition systems are *complete trace equivalent* iff their initial states are completed trace equivalent.

Note that our notion of completed trace equivalence has as an additional constraint that the set of traces should also be equivalent. This is essential for distinguishing systems with infinite, non terminating behaviour. Take as example an a-loop and a b-loop. The observable behaviour of these two transition systems is very different, but their sets of completed traces are both empty. By including the sets of traces in the definition of completed trace equivalence they are distinguished again.

Consider the labelled transition systems depicted in figure 2.6. They are trace equivalent. They are also completed trace equivalent since their completed trace sets are both empty. However, when it would be possible to block the $reset$ action, they are not completed trace equivalent any more. Blocking the $reset$ transition boils down to removing $reset$ transitions. Then the transition system at the left has set as a completed trace, which the transition system at the right does not have. Completed trace equivalence is not compositional, i.e., it can be jeopardised when equivalent systems are placed in a context that can influence their behaviour. Compositionality, also called congruence, is a much desired property and is the cornerstone of the algebraic approach to system description (see chapter 4). Fortunately, we have good alternatives for language and completed trace equivalence that are compositional, which will be introduced in the next sections.

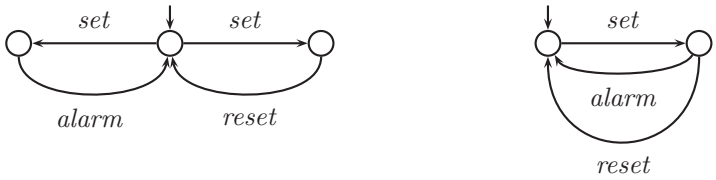


Figure 2.6: Two completed trace equivalent alarm clocks

2.3.3 ★Failures equivalence

The equivalence that is closest to completed trace equivalence and that is a congruence when blocking of actions allowed, is *failures equivalence*. The typical property of failure equivalence is that it relates as many behaviours as possible, while preserving traces and deadlocks, even if the behaviours are placed in an environment constructed of common process operators (see chapters 4 and 5).

The definition of failures equivalence has two steps. First a refusal set of a state t is defined to contain those actions that cannot be performed in t. Then a failure pair is defined to be a trace ending in some refusal set.

Definition 2.3.5 (Failures equivalence). Let $A = (S, Act, \longrightarrow, s, T)$ be a labelled transition system. A set $F \subseteq Act \cup \{\checkmark\}$ is called a *refusal set* of a state $t \in S$,

- if for all actions $a \in F$ there is no $t' \in S$ such that $t \xrightarrow{a} t'$, and
- if $\checkmark \in F$, then $t \notin T$, i.e., t cannot terminate.

The *set of failure pairs*, $FailurePairs(t)$, of a state $t \in S$ is inductively defined as follows

- $(\epsilon, F) \in FailurePairs(t)$ if F is a refusal set of t.
- $(\checkmark, F) \in FailurePairs(t)$ iff $t \in T$ and F is a refusal set of t, and
- If $t \xrightarrow{a} t'$ and $(\sigma, F) \in FailurePairs(t')$ then $(a\sigma, F) \in FailurePairs(t)$.

Two states $t, u \in S$ are *failures equivalent* iff $FailurePairs(t) = FailurePairs(u)$. Two transition systems are *failures equivalent* iff their initial states are failures equivalent.

The labelled transition systems depicted in figure 2.7 are failures equivalent. Initial states of both labelled transition systems refuse the set $\{alarm, reset\}$; hence $(\epsilon, \{alarm, reset\})$ is a failure pair for both initial states. The lower state in the figure at the right has, for instance, failure pairs $(set, \{set\})$ and $(set, \emptyset)$. But these are also failure pairs of the left and right states of both figures. These transitions systems show that behaviour demonstrating a lack of choice at the left is equal to behaviour at the right with the possibility to choose between a $reset$, or an $alarm$ action. We say that failures equivalence does not preserve the branching structure of behaviour.

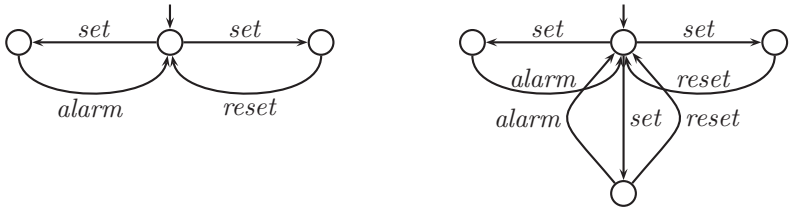
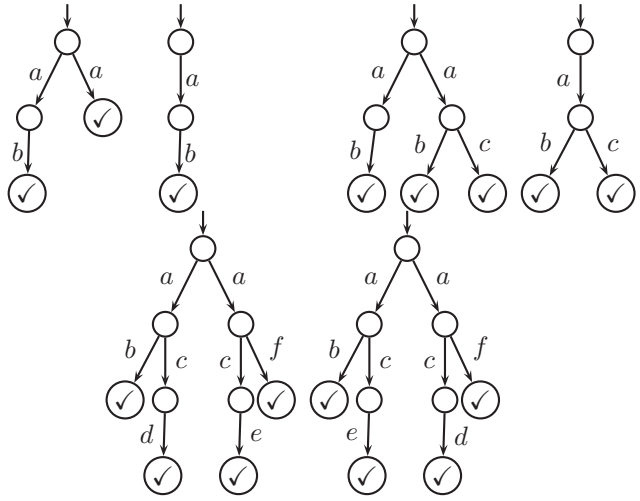


Figure 2.7: Two failure-equivalent alarm clocks

Exercise 2.3.6. State whether the following pairs of transition systems are language and/or failures equivalent.



2.3.4 Strong bisimulation equivalence

Bisimulation equivalence, also referred to as strong bisimulation equivalence, or strong bisimilarity is the most important process equivalence although its definition is far more complex than trace equivalence.

One reason for its importance is that if two processes are bisimulation equivalent, they cannot be distinguished by any realistic form of behavioural observation. This includes for instance observations where performed actions can be undone, where copies of the system under observation are made which are tested separately, where it can be observed that actions are not possible and where it can be observed that arbitrarily long sequences of actions are possible in a certain state. Hence, if two processes are bisimilar they can certainly be considered indistinguishable under observation.

Another reason is that the algorithms for checking bisimulation equivalence are efficient, contrary to the algorithms for checking any equivalence based on a form of traces, which are generally PSPACE-complete or worse.

The idea behind bisimulation is that two states are related if the actions that can be done in one state, can be done in the other, too. We say that the second state simulates the first. Moreover, if one action is simulated by another, the resulting states must be related also.

Definition 2.3.7 (Bisimulation). Let $A{=}(S, Act, \longrightarrow, s, T)$ be a labelled transition system. A binary relation $R \subseteq S \times S$ is called a *strong bisimulation relation* iff for all $s, t \in S$ such that sRt holds, it also holds for all actions $a \in Act$ that:

1. if $s \xrightarrow{a} s'$, then there is a $t' \in S$ such that $t \xrightarrow{a} t'$ with $s'Rt'$,
2. if $t \xrightarrow{a} t'$, then there is a $s' \in S$ such that $s \xrightarrow{a} s'$ with $s'Rt'$, and
3. $s \in T$ if and only if $t \in T$.

Two states s and t are *strongly bisimilar*, denoted by $s \leftrightarrows t$, iff there is a strong bisimulation relation R such that sRt. Two labelled transition systems are *strongly bisimilar* iff their initial states are bisimilar.

Often the adjective *strong* is dropped, speaking about bisimulation rather than strong bisimulation. However, we will see several other variants of bisimulation and in those cases the use of 'strong' helps us stress the difference.

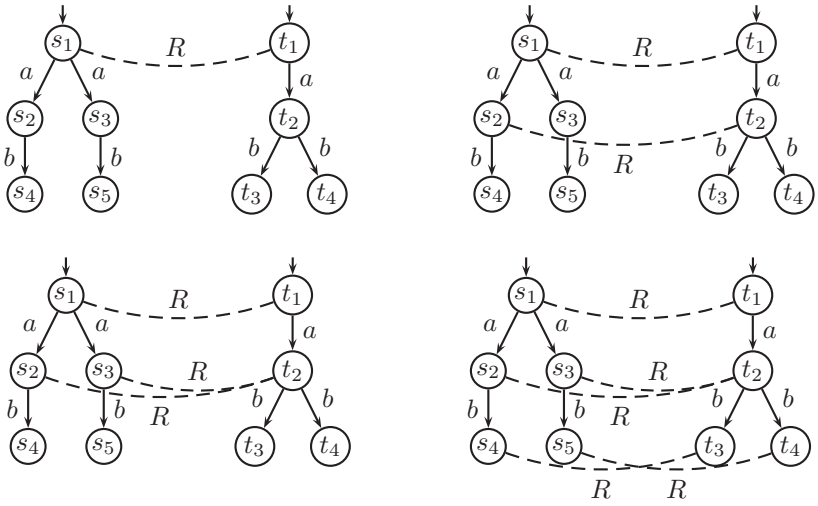


Figure 2.8: Showing two LTSs bisimilar

There are several techniques to show that two labelled transition system are bisimilar. Computer algorithms generally use partition refinement for the Relation Coarsest Partitioning problem [111, 146]. For small transition systems a more straightforward technique is generally adequate. Consider the transition systems in figure 2.8. In order to show that the initial states s_1 and t_1 are bisimilar, a bisimulation relation R must be constructed to relate these two states. We assume that this can be done. Hence, we draw an arc between s_1 and t_1 and label it with R. If R is a bisimulation, then every transition from s_1 must be mimicked by a similarly labelled transition from t_1. More concretely, the a-transition from s_1 to s_2 can only be mimicked by an a-transition from t_1 to t_2. So, s_2 and t_2 must be related, too. We also draw an arc to indicate this (see the second picture in figure 2.8). Now we can proceed by showing that the transition from s_1 to s_3 must also be mimicked by the a-transition from t_1 to t_2. Hence, s_3 is related to t_2 (see the third picture). As a rule of thumb, it generally pays of to first choose a transition from the side with more choices in order to force the other side to perform a certain transition. Otherwise, there might be a choice, and several possibilities need to be considered. E.g., the a-transition t_1 to t_2 can be simulated by either the transition from s_1 to s_2, or the one from s_1 to s_3.

The relation R needs to be extended to all reachable nodes. Therefore, we consider the relation between s_2 and t_2. We continue the process sketched before, but now let the transitions from the right transition system be simulated by the left one, because

the states s_2 and s_3 are deterministic. The relation R is extended as indicated in the fourth picture of figure 2.8. It needs to be checked that all related states satisfy all the requirements in definition 2.3.7. As this is the case, R is a bisimulation relation, the initial states are bisimilar and therefore the systems are bisimilar.

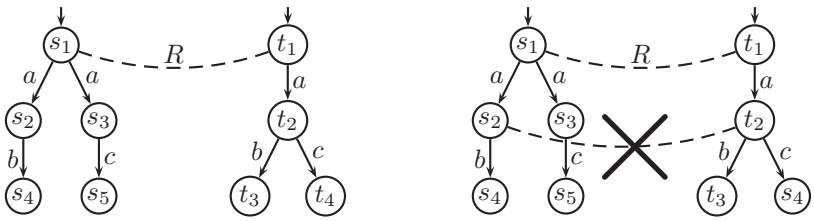


Figure 2.9: Two non-bisimilar labelled transition systems

Now consider the transition systems in figure 2.9. There are three actions a, b and c. These two transition systems are not bisimilar.

Before showing this formally, we first give an intuitive argument why these two processes are different. Let actions a, b and c stand for pressing one of three buttons. If a transition is possible, the corresponding button can be pressed. If a transition is not possible, the button is blocked.

Now suppose a customer ordered the transition system at the right (with initial state t_1) and a 'malicious' supplier delivered a box with the behaviour of the transition system at the left. If the customer cannot experience the difference, the supplier did an adequate job. However, the customer can first press an a button such that the box ends up being in state s_3. Now the customer, thinking that she is in state t_2 expects that both b and c can be pressed. She, however, finds out that b is blocked, from which she can conclude that she has been deceived and has an argument to sue the supplier.

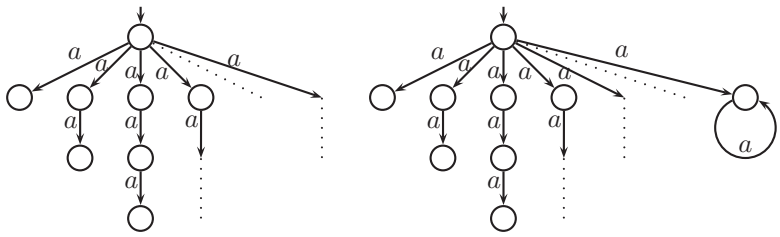
Now note that in both behaviours in figure 2.9 the same sequence of actions can be performed, namely $a\,b$ and $a\,c$; in other words, they are (completed) trace equivalent. Yet, the behaviour of both systems can be experienced to be different!

If one tries to show both transition systems bisimilar using the method outlined above, then in the same way as before, state s_2 must be related to state t_2. However, a c transition is possible from state t_2 that cannot be mimicked by state s_2 which has no outgoing c transition. Hence, s_2 cannot be related to t_2 and consequently, s_1 cannot be bisimilar to t_1.

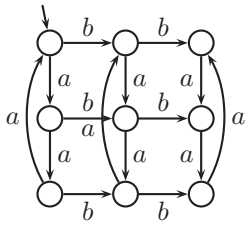
A pleasant property of bisimulation is that for any labelled transition system, there is a unique minimal transition system (up to graph isomorphism) which is bisimilar to it.

Exercise 2.3.8. State for each pair of transition systems from exercise 2.3.2 whether they are bisimilar.

Exercise 2.3.9. Show that the following transition systems are not bisimilar, where the transition system to the left consists of sequences of a-transitions with length n for each $n \in \mathbb{N}$. The transition system to the right is the same except that it can additionally do an infinite sequence of a-transitions.



Exercise 2.3.10. Give the unique minimal labelled transition system that is bisimilar to the following one:



2.3.5 The Van Glabbeek linear time – branching time spectrum

As stated before, there is a myriad of process equivalences. A nice classification of some of these has been given by Van Glabbeek[2] [71]. He produced the so-called linear time-branching time spectrum of which a part is depicted in figure 2.10. At the top the finest equivalence, relating the fewest states, is depicted and at the bottom the coarsest equivalence, relating the most states, is found. The arrows indicate that an equivalence is strictly coarser. For example, if processes are bisimulation equivalent, then they are also 2-nested simulation equivalent. Bisimulation equivalence is the finest equivalence and trace equivalence the coarsest. If two processes are bisimilar then they are equivalent with respect to all equivalences depicted in the spectrum. If processes are not bisimilar, then it makes sense to investigate whether they are equal with respect to another equivalence.

Each equivalence has its own properties, and it goes too far to treat them all. As an illustration we relate a few types of observations to some equivalences. Suppose that we can interact with a machine that is equipped with an undo button. So, after doing some actions, we can go back to where we came from. Then one can devise tests that precisely distinguish between processes that are not ready simulation equivalent. So, ready simulation is tightly connected to the capability of undoing actions. In a similar way, possible futures equivalence is strongly connected to the capability of predicting which actions are possible in the future and 2-nested simulation equivalence combines them both.

The Van Glabbeek spectrum owes its existence to nondeterminism. If transition systems are deterministic then the whole spectrum collapses. In that case two states are

[2]Rob van Glabbeek (1960-. . .) classified thousands of process equivalences. He contributed to various areas in mathematics and theoretical computer science, including concurrency theory, linear logic and protocols for wireless networks. He ran for the Dutch parliament in 2012 for the Libertarian Party.

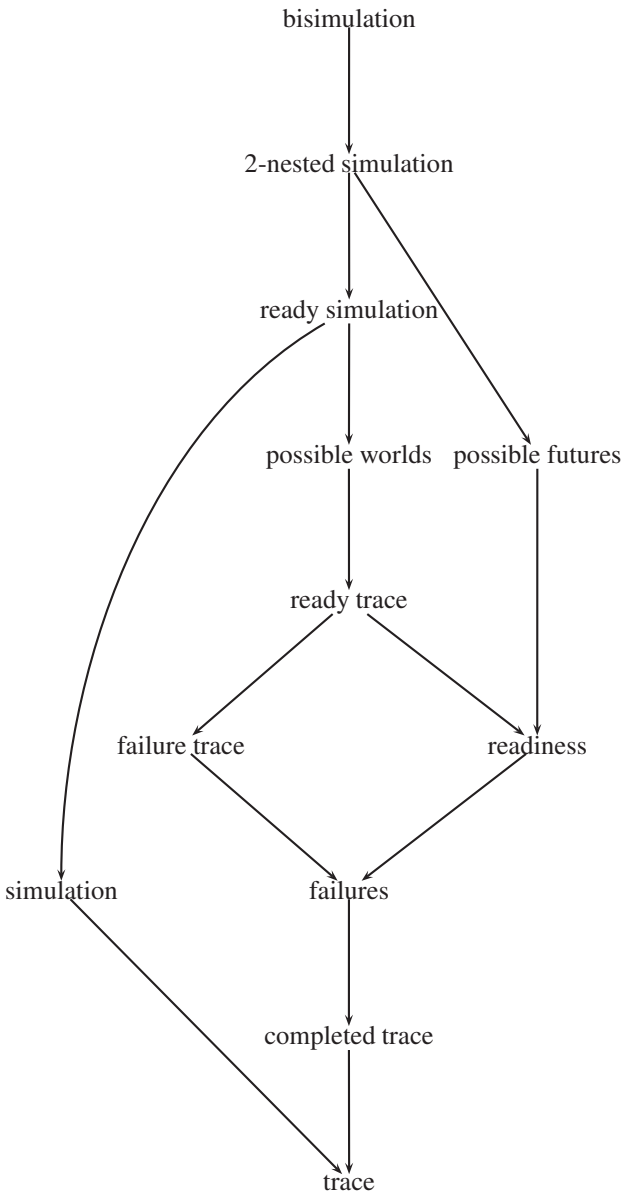


Figure 2.10: The Van Glabbeek linear time – branching time spectrum

bisimulation equivalent if and only if they are trace equivalent. We state this theorem precisely here and provide the proof as an example of how properties of bisimulation are proven.

Definition 2.3.11. We call a labelled transition system $A = (S, Act, \longrightarrow, s, T)$ *deterministic* iff for all reachable states $t, t', t'' \in S$ and action $a \in Act$ it holds that if $t \xrightarrow{a} t'$ and $t \xrightarrow{a} t''$ then $t' = t''$.

Theorem 2.3.12. Let $A = (S, Act, \longrightarrow, s, T)$ be a deterministic transition system. For all states $t, t' \in S$ it holds that

$$Traces(t) = Traces(t') \quad \text{iff} \quad t \Leftrightarrow t'.$$

Proof. We only prove the case from left to right. The proof from right to left is much easier.

In order to show that $t \Leftrightarrow t'$, we need to show the existence of a bisimulation relation R such that tRt'. We coin the following relation for all states $u, u' \in S$:

$$u \, R \, u' \quad \text{iff} \quad Traces(u) = Traces(u').$$

Finding the right relation R is generally the crux in such proofs. Note that R is indeed suitable, as R relates t and t'.

So, we are only left with showing that R is indeed a bisimulation relation. This boils down to checking the properties in definition 2.3.7 (strong bisimulation). Assume that for states u and v we have uRv. Then

1. Suppose $u \xrightarrow{a} u'$. According to definition 2.3.1 (trace equivalence) $a\sigma \in Traces(u)$ for all traces $\sigma \in Traces(u')$. Furthermore, as $Traces(u) = Traces(v)$, it holds that $a\sigma \in Traces(v)$ or in other words, $v \xrightarrow{a} v'$ for some state $v' \in S$. We are left to show that $u'Rv'$, or in other words:

   $$Traces(u') = Traces(v').$$

   We prove this by mutual set inclusion, restricting ourselves to only one case, as both are almost identical. So, we prove $Traces(u') \subseteq Traces(v')$. Consider a trace $\sigma \in Traces(u')$. It holds that $a\sigma \in Traces(u)$, and consequently $a\sigma \in Traces(v)$. Hence, there is a v'' such that $v \xrightarrow{a} v''$ and $\sigma \in Traces(v'')$. As the transition system A is deterministic, $v \xrightarrow{a} v'$ and $v \xrightarrow{a} v''$, we can conclude $v' = v''$. Ergo, $\sigma \in Traces(v')$.

2. This case is symmetric to the first case and is therefore omitted.

3. If $u \in T$, then $\checkmark \in Traces(u)$. As u and v are related, it follows by definition of R that $\checkmark \in Traces(v)$. So, $v \in T$. Similarly, it can be shown that if $v \in T$ then $u \in T$ must hold. □

Exercise 2.3.13. Prove that bisimilarity on a given labelled transition system is an equivalence relation, i.e., it is reflexive ($s \Leftrightarrow s$ for any $s \in S$), symmetric (if $s \Leftrightarrow t$ then $t \Leftrightarrow s$ for all states s and t) and transitive (if $s \Leftrightarrow t$ and $t \Leftrightarrow u$, then $s \Leftrightarrow u$ for all states s, t, u).

Figure 2.11: The internal action τ is not visible

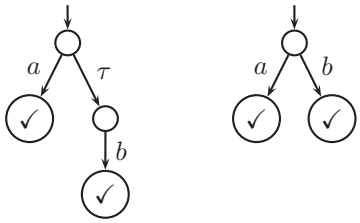


Figure 2.12: The internal action τ is indirectly visible

2.4 Behavioural abstraction

Although the examples given hitherto may give a different impression, the behaviour of systems can be utterly complex. The only way to obtain insight in such behaviour is to use abstraction. The most common abstraction mechanism is to declare an action as internal and hence, unobservable and adapt the notions of equivalence such that the unobservable nature of such internal actions is taken into account.

2.4.1 The internal action τ

We say that an action is internal, if we have no way of observing it directly. We use the special symbol τ to denote internal actions collectively. We generally assume that internal action is available in every labelled transition system, i.e., $\tau \in Act$. Typical for an internal action is that if it follows another action, it is impossible to say whether it is there. So, the transition systems in figure 2.11 cannot be distinguished, because the τ after the a cannot be observed. Such internal actions are called *inert*.

However, in certain cases the presence of an internal action can be indirectly observed, although the action by itself cannot be seen. Suppose one expects the behaviour of the transition system at the right of figure 2.12. It is always possible to do an a-action, as long as neither an a nor a b have been done. Now suppose the actual behaviour is that of the transition system at the left and one insists on doing an action a. If the internal action silently happens, a deadlock is observed, because it is impossible to do action a anymore. Hence, when actions can be chosen and deadlocks observed, it can be determined by observations only that the behaviour at the left is not the same as that of the transition system at the right.

With the internal action present, equivalences for processes must take into account that we cannot observe the internal action directly. Next, the most important of such equivalences are given.

2.4.2 Weak trace equivalence

Weak traces are obtained by absorbing the internal action in a trace. This is the natural notion if we can observe all but the internal action, and we cannot interact with the system or observe that it is in a deadlock. Formally, two processes are weak trace equivalent, if their sets of weak traces, i.e., traces in which τ-transitions are ignored, are the same.

Definition 2.4.1 (Weak trace equivalence). Let $A = (S, Act, \longrightarrow, s, T)$ be a labelled transition system. The set of *weak traces* $WTraces(t)$ for a state $t \in S$ is the minimal set satisfying:

1. $\epsilon \in WTraces(t)$.
2. $\checkmark \in WTraces(t)$ iff $t \in T$, and
3. if there is a state $t' \in S$ such that $t \xrightarrow{a} t'$ ($a \neq \tau$) and $\sigma \in WTraces(t')$ then $a\sigma \in WTraces(t)$.
4. if there is a state $t' \in S$ such that $t \xrightarrow{\tau} t'$ and $\sigma \in WTraces(t')$ then $\sigma \in WTraces(t)$.

Two states $t, u \in S$ are called *weak trace equivalent* iff $WTraces(t) = WTraces(u)$. Two transition systems are *weak trace equivalent* iff their initial states are weak trace equivalent.

Weak trace equivalence is the weakest of all behavioural equivalences. It does not preserve deadlocks, nor any other branching behaviour. Weak trace equivalence is hard to calculate for a given transition system and it is much harder to obtain a smallest transition system preserving weak trace equivalence. There is in general no unique smallest transition system modulo weak trace equivalence.

But in practice transition systems modulo weak trace equivalence can be much smaller than those obtained with any other equivalence. And although one should always be aware of the properties not preserved by weak trace equivalence, those small transition systems expose behaviour more clearly which is of great value for our understanding.

2.4.3 (Rooted) Branching bisimulation

The definition of branching bisimulation is very similar to that of strong bisimulation. But now, instead of letting a single action be simulated by a single action, an action can be simulated by a sequence of internal transitions, followed by that single action. See the diagram at the left of figure 2.13. It can be shown that all states that are visited via the τ-actions in this diagram are branching bisimilar.

If the action to be simulated is a τ, then it can be simulated by any number of internal transitions, or even by no transition at all, as the diagram in the middle of figure 2.13 shows.

If a state can terminate, it does not need to be related to a terminating state. It suffices if a terminating state can be reached after a number of internal transitions, as shown at the right of figure 2.13.

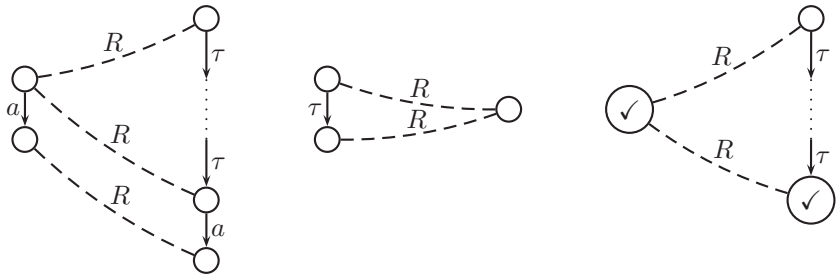


Figure 2.13: Branching bisimulation

Definition 2.4.2 (Branching bisimulation). Consider the labelled transition system $A = (S, Act, \longrightarrow, s, T)$. We call a relation $R \subseteq S \times S$ a *branching bisimulation relation* iff for all $s, t \in S$ such that sRt, the following conditions hold for all actions $a \in Act$:

1. If $s \xrightarrow{a} s'$, then
   - either $a = \tau$ and $s'Rt$, or
   - there is a sequence $t \xrightarrow{\tau} \cdots \xrightarrow{\tau} t'$ of (zero or more) τ-transitions such that sRt' and $t' \xrightarrow{a} t''$ with $s'Rt''$.
2. Symmetrically, if $t \xrightarrow{a} t'$, then
   - either $a = \tau$ and sRt', or
   - there is a sequence $s \xrightarrow{\tau} \cdots \xrightarrow{\tau} s'$ of (zero or more) τ-transitions such that $s'Rt$ and $s' \xrightarrow{a} s''$ with $s''Rt'$.
3. If $s \in T$, then there is a sequence of (zero or more) τ-transitions $t \xrightarrow{\tau} \cdots \xrightarrow{\tau} t'$ such that sRt' and $t' \in T$.
4. Again, symmetrically, if $t \in T$, then there is a sequence of (zero or more) τ-transitions $s \xrightarrow{\tau} \cdots \xrightarrow{\tau} s'$ such that $s'Rt$ and $s' \in T$.

Two states s and t are *branching bisimilar*, denoted by $s \leftrightarrows_b t$, if there is a branching bisimulation relation R such that sRt. Two labelled transition systems are *branching bisimilar* if their initial states are branching bisimilar.

Example 2.4.3. In figure 2.14 two transition systems are depicted. We can determine that they are branching bisimilar in the same way as for strong bisimulation. So, first assume that the initial states must be related, via some relation R. For R to be a branching bisimulation, the transition $s_1 \xrightarrow{a} s_3$ must be mimicked. This can only be done by two transitions $t_1 \xrightarrow{\tau} t_3 \xrightarrow{a} t_4$. So, as depicted in the second diagram, s_1 must be related to the intermediate state t_3 and s_3 must be related to t_4. Now, by letting the transition $t_1 \xrightarrow{b} t_2$ be simulated by $s_1 \xrightarrow{\tau} s_2 \xrightarrow{b} s_5$ the relation is extended as indicated in the third diagram. Ultimately, the relation R must be extended as indicated

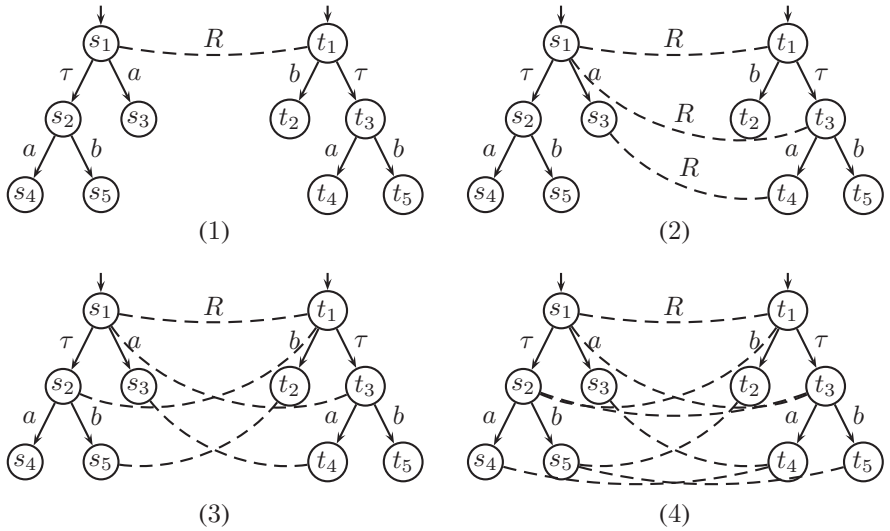


Figure 2.14: Two branching bisimilar transition systems

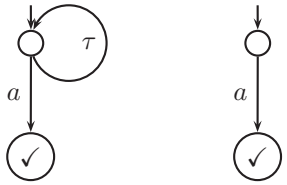


Figure 2.15: Branching bisimulation does not preserve τ-loops

in the fourth diagram. It requires a careful check that this relation is indeed a branching bisimulation relation.

Branching bisimulation equivalence has a built-in notion of *fairness*. That is, if a τ-loop exists, then no infinite execution sequence will remain in this τ-loop forever if there is a possibility to leave it. The intuition is that there is zero chance that any exit from the τ-loop will ever be chosen. It is straightforward to show that the initial states in the two labelled transition systems in figure 2.15 are branching bisimilar.

A state with a τ-loop is also called a *divergent* state. There are times when it is desired to distinguish divergent states from non-divergent states. This distinction can be achieved by requiring that a branching bisimulation relation has the following additional property.

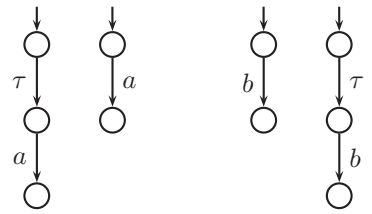
- If sRt then there is an infinite sequence $s \xrightarrow{\tau}\xrightarrow{\tau} \cdots$ iff there is an infinite sequence $t \xrightarrow{\tau}\xrightarrow{\tau} \cdots$.

We call such a branching bisimulation relation a divergence preserving branching bisimulation relation. Two states s and t are *divergence preserving branching bisimilar*,

notation $s \leftrightarrows_{db} t$, iff there is a divergence preserving branching bisimulation R relating them. The initial states of the transition systems in figure 2.15 are not divergence preserving branching bisimilar.

Branching bisimulation (with or without divergence) has an unpleasant property, namely, it is not compositional. If an alternative is added to the initial state then the resulting processes may cease to be bisimilar. This is illustrated in the following example. In chapter 4, we will see that adding an alternative to the initial state is a common operation.

Example 2.4.4. Consider the following two pairs of labelled transition systems.



These labelled transition systems are branching bisimilar. However, their pairwise compositions with a choice at their initial states, depicted in figure 2.16, are not branching bisimilar. For assume that they were branching bisimilar, then there should be a branching bisimulation relation relating their initial states. Since the labelled transition system at the right affords an a-transition from the initial state, the labelled transition system at the left should mimic it by performing a τ-transition followed by an a-transition. This means that the intermediate state after performing the τ-transition and before performing the a-transition should be related by the same relation to the initial state of the right-hand-side labelled transition system, as depicted in the lower part of figure 2.16. But this is impossible since the initial state of labelled transition system at the right allows a b-transition, while the related state at the left cannot mimic this b transition.

The problem explained in example 2.4.4 (and depicted in figure 2.16) is known as "the rootedness problem". It is caused because doing a τ means that the option to do an observable action (e.g., a) disappears. Milner [135] showed that this problem can be overcome by adding a rootedness condition: two processes are considered equivalent if they can simulate each other's initial transitions (including τ-transitions), such that the resulting processes are branching bisimilar. This leads to the notion of *rooted branching bisimulation*, which is presented below.

Definition 2.4.5 (Rooted branching bisimulation). Let $A = (S, Act, \longrightarrow, s, T)$ be a labelled transition system. A relation $R \subseteq S \times S$ is called a *rooted branching bisimulation relation* iff it is a branching bisimulation relation and it satisfies for all $s \in S$ and $t \in S$ such that sRt:

1. if $s \xrightarrow{a} s'$, then there is a $t' \in S$ such that $t \xrightarrow{a} t'$ and $s' \leftrightarrows_b t'$,
2. symmetrically, if $t \xrightarrow{a} t'$, then there is an $s' \in S$ such that $s \xrightarrow{a} s'$ and $s' \leftrightarrows_b t'$.

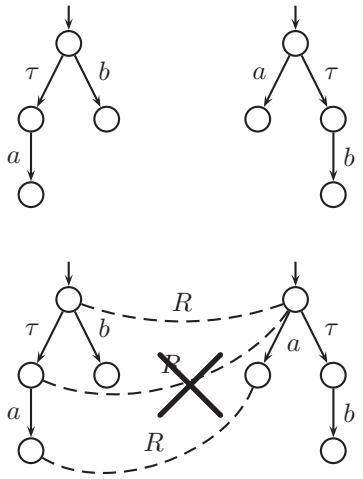


Figure 2.16: Root problem in branching bisimulation

Two states $s \in S$ and $t \in S$ are *rooted branching bisimilar*, denoted by $s \leftrightsquigarrow_{rb} t$, iff there is a rooted branching bisimulation relation R such that sRt. Two transition systems are *rooted branching bisimilar* iff their initial states are rooted branching bisimilar.

Rooted divergence sensitive branching bisimulation can be defined in exactly the same way. We write $s \leftrightsquigarrow_{rdb} t$ to express that states s and t are rooted divergence sensitive bisimilar.

Branching bisimulation equivalence strictly includes rooted branching bisimulation equivalence, which in turn strictly includes bisimulation equivalence. A similar set of strict inclusions can be given for divergence preserving bisimulation.

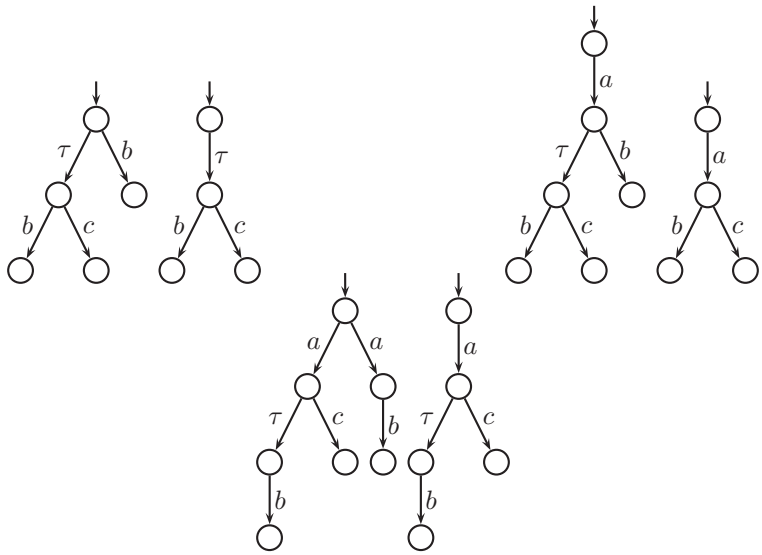
$$\leftrightarrow \subset \leftrightsquigarrow_{rb} \subset \leftrightsquigarrow_{b}, \qquad \leftrightarrow \subset \leftrightsquigarrow_{rdb} \subset \leftrightsquigarrow_{db} \subset \leftrightsquigarrow_{b} .$$

Note that in the absence of τ, strong bisimulation and all variants of branching bisimulation coincide.

When behaviours such as those in figure 2.11 are equivalent, which is generally accepted as a reasonable minimal requirement for internal actions, and when processes can be put in parallel, rooted branching bisimulation is the natural finest equivalence. The argument for this goes beyond the scope of this chapter, but it stresses that branching bisimulation is a very natural notion.

Exercise 2.4.6. Show using the definition of rooted branching bisimulation that the two labelled transition systems in figure 2.11 are rooted branching bisimilar. Show also that the two transition systems in figure 2.12 are neither rooted branching bisimilar nor branching bisimilar.

Exercise 2.4.7. Which of the following pairs of transition systems are branching and/or rooted branching bisimilar.



Exercise 2.4.8. With regard to the examples in exercise 2.4.7 which τ-transitions are *inert* with respect to branching bisimulation, i.e., for which τ-transitions $s \xrightarrow{\tau} s'$ are the states s and s' branching bisimilar?

2.4.4 ★(Rooted) Weak bisimulation

A slight variation of branching bisimulation is weak bisimulation. We give its definition here, because weak bisimulation was defined well before branching bisimulation was invented and therefore weak bisimulation is much more commonly used in the literature.

The primary difference between branching and weak bisimulation is that branching bisimulation preserves 'the branching structure' of processes. For instance the last pair of transition systems in exercise 2.4.7 are weakly bisimilar, although the initial a in the transition system at the left can make a choice that cannot be mimicked in the transition system at the right. The branching structure is not respected.

It is useful to know that (rooted) branching bisimilar processes are also (rooted) weakly bisimilar. Furthermore, from a practical perspective, it hardly ever matters whether branching or weak bisimulation is used, except that the algorithms to calculate branching bisimulation on large graphs are more efficient than those for weak bisimulation.

Definition 2.4.9 (Weak bisimulation). Consider the labelled transition system $A = (S, Act, \longrightarrow, s, T)$. We call a relation $R \subseteq S \times S$ a *weak bisimulation relation* iff for all $s, t \in S$ such that sRt, the following conditions hold:

1. If $s \xrightarrow{a} s'$, then
   - either $a = \tau$ and $s'Rt$, or
   - there is a sequence $t \xrightarrow{\tau} \cdots \xrightarrow{\tau} \xrightarrow{a} \xrightarrow{\tau} \cdots \xrightarrow{\tau} t'$ such that $s'Rt'$.
2. Symmetrically, if $t \xrightarrow{a} t'$, then

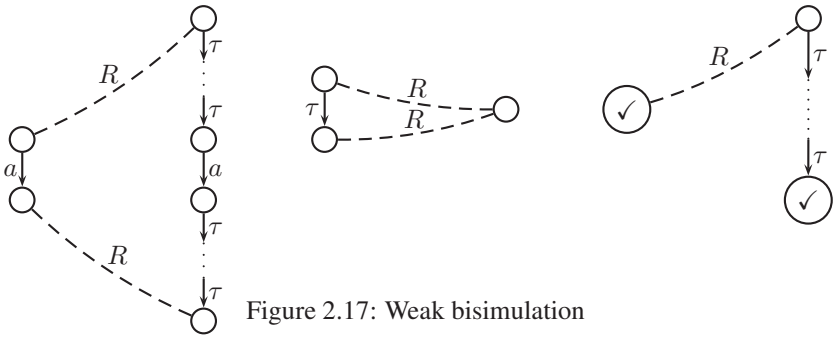


Figure 2.17: Weak bisimulation

- either $a = \tau$ and sRt', or
- there is a sequence $s \xrightarrow{\tau} \cdots \xrightarrow{\tau} \xrightarrow{a} \xrightarrow{\tau} \cdots \xrightarrow{\tau} s'$ such that $s'Rt'$.

3. If $s \in T$, then there is a sequence $t \xrightarrow{\tau} \cdots \xrightarrow{\tau} t'$ such that $t' \in T$.
4. Again, symmetrically, if $t \in T$, then there is a sequence $s \xrightarrow{\tau} \cdots \xrightarrow{\tau} s'$ such that $s' \in T$.

Two states s and t are *weakly bisimilar*, denoted by $s \leftrightarrows_w t$, iff there is a weak bisimulation relation R such that sRt. Two labelled transition systems are *weakly bisimilar* iff their initial states are weakly bisimilar.

In figure 2.17 weak bisimulation is illustrated. Compare this figure with figure 2.13 for branching bisimulation. Note that weak bisimulation is more relaxed in the sense that the weak bisimulation relation R does not have to relate that many states.

The notion of rooted weak bisimulation is defined along the same lines as rooted branching bisimulation. The underlying motivation is exactly the same.

Definition 2.4.10 (Rooted weak bisimulation). Let $A = (S, Act, \longrightarrow, s, T)$ be a labelled transition system. A relation $R \subseteq S \times S$ is called a *rooted weak bisimulation relation* iff R is a weak bisimulation relation and it satisfies for all $s, t \in S$ such that sRt:

1. if $s \xrightarrow{\tau} s'$, then there is a sequence $t \xrightarrow{\tau} \xrightarrow{\tau} \cdots \xrightarrow{\tau} t'$ of at least length 1 and $s' \leftrightarrows_w t'$, and
2. symmetrically, if $t \xrightarrow{\tau} t'$, then there is sequence of at least length 1 of τ-steps $s \xrightarrow{\tau} \xrightarrow{\tau} \cdots \xrightarrow{\tau} s'$ and $s' \leftrightarrows_w t'$.

Two states $s \in S$ and $t \in S$ are *rooted weakly bisimilar*, denoted by $s \leftrightarrows_{rw} t$, iff there is a rooted weak bisimulation relation R such that sRt. Two transition systems are *rooted weakly bisimilar* iff their initial states are rooted weakly bisimilar.

We finish this section by showing the relationships between weak and branching bisimulation, where $\subset$ denotes strict set inclusion.

$$\leftrightarrows \subset \leftrightarrows_{rb} \subset \leftrightarrows_b \subset \leftrightarrows_w, \qquad \leftrightarrows \subset \leftrightarrows_{rb} \subset \leftrightarrows_{rw} \subset \leftrightarrows_w .$$

Note that rooted weak bisimulation and branching bisimulation are incomparable. Note also that we can define divergence preserving weak bisimulation, but as this notion is hardly used and its definition is exactly the same as that for divergence preserving branching bisimulation, we do not do this explicitly here.

Exercise 2.4.11. Which of the pairs of transition systems of figures 2.11 and 2.12 are (rooted) weakly bisimilar.

Exercise 2.4.12. Which of the pairs of transition systems of exercise 2.4.7 are (rooted) weakly bisimilar. Which τ-transitions are inert with respect to weak bisimulation (cf., exercise 2.4.8).

Exercise 2.4.13. Prove that branching bisimulation is a weak bisimulation relation.

2.5 Historical notes

State machines have been used historically to describe programs. They already occur in Turing Machines to formalise the concept of computation. Subsequently, they were heavily used in language theory, especially for compiler construction [5].

The idea of capturing the abstract behaviour of computer programs by automata has been the corner-stone of the operational approach to semantics (meaning of programs) advocated by McCarthy [129] and Plotkin [151].

However, adding the notion of interaction with the environment was required to develop an abstract theory of system behaviour, which was proposed by pioneers such as Petri [150], Bekič [24] and Milner [135]. This led to various theories of process calculi, exemplified by CCS [135], CSP [101], ACP [18] and an early formal and standardised behavioural specification language LOTOS [106]. We refer to [11, 54] for more detailed historical accounts and to [3, 12, 159] for excellent textbook introductions to the field.

Regarding fundamental models of behaviour, there are various alternatives to labelled transition systems. Instead of labelled transition systems, one can use sets of traces, if necessary decorated to represent parallel behaviour (Mazurkiewicz traces) [49] and even sets representing behavioural trees. A fundamental problem with such traces and trees is that sets cannot contain themselves (axiom of foundation). So, a loop cannot be represented in such sets as this would require a set that contains itself. This led to the use of projective limit models and metric spaces where the existence of an object representing a loop could be proven to exist [20]. It was also the motivation to start the work on ACP [26] and led to research into non-well-founded sets [4].

Rather independently and originally invented to describe chemical processes, Petri nets were developed which can be viewed as a higher level description of automata, especially suitable to describe data processing [107], and as such becoming a basis for the description of business information systems [1].

Partly in reaction to the atomic nature of actions as proposed by Milner, a field baptised ‘true concurrency semantics’ arose stating that events are not atomic and should therefore not be treated as such. The history of the term goes back to 1988 (and possibly before) were a group of prominent researchers on concurrency theory got together

for a workshop in Königswinter, Germany. We cite the following interesting anecdote from a report of this meeting [154]:

> Partial order semantics were popular at the conference (Königswinter 1988). Some of its adherents have taken to calling p.o. [(partial order)] semantics true concurrency, prompting Robin Milner to start his talk by speaking up for "false concurrency".

Event structures [178] are a typical instance of models with a true concurrency semantics. Some regard Petri nets to fall into this category as well [139]. In response Milner referred to process calculi as 'false concurrency'.

The idea that two objects are the same if they have exactly the same properties is rather old, and is sometimes referred to as Leibniz equality. In the context of behaviour this is translated to the statement that two systems are equal if the difference cannot be observed. Although self evident in retrospect, it came as a surprise that there are very many different ways to observe processes. This led to a myriad of behavioural equivalences, of which [70, 71] provide compact overviews, in the settings with and without internal actions, respectively. The notion of strong bisimulation was first proposed by Park [148] and Milner [133], which was inspired by earlier automata-theoretic notions, which were in turn explored by Milner and adopted in this context. The same notion has been developed in various other contexts, including in the context of modal logics and non-well-founded sets, of which [162] gives a detailed historical account and [161] provides an excellent textbook introduction. The notion of internal action and weak bisimulation was proposed by Milner [133]. Branching bisimulation was defined by Van Glabbeek and Weijland [69, 72].