Notes on Labelled Transition Systems and Bisimulation

Prakash Panangaden[*]

11th May 2020

1 Introduction

We are interested in modelling systems and studying their behaviour. In particular, we would like to know when two systems have the same behaviour. So far, we have been looking at automata as language recognizers, but now we would like to think of them as models of systems.

The intuition is as follows. We imagine that we have a "black box" which has some internal states; *we cannot see these states.* The box is equipped with some buttons which we are free to press at any point. When we press a button it either goes down (the action is accepted) or the button refuses to go down (the action is rejected). We can perform experiments and see whether an action is accepted or not. If the action is accepted then a transition occurs, but we cannot see exactly what happens. We are interested in the behaviour of the system in the sense of knowing how it responds to our button-pushing experiments.

To fix ideas consider the following example shown in Fig. 1. We have a vending machine that dispenses coffee and tea. It has a slot for inserting dollar coins and two buttons: one labelled "tea" and the other labelled "coffee." When you arrive you can try to press the "tea" button but it refuses to respond; it is waiting for you to put money in. You insert a dollar and find that this action is accepted. Now you can try various things: inserting another dollar or pushing one of the buttons. If you insert another

[*]Based on an ealrier version written with help from Maja Frydrychowicz and Andrew Schaaf

$$\begin{tikzcd}
& s_0 \arrow[d, "\mathsf{m}"'] & \\
& s_1 \arrow[dl, "\mathsf{c}"'] \arrow[dr, "\mathsf{t}"] & \\
s_2 & & s_3
\end{tikzcd}$$

Figure 1: A vending machine LTS

dollar it just refuses to go in (real machines will probably have a coin return slot and return the coin). If you press the "tea" button the action is accepted now and the machine dispenses tea; if you were to press the coffee button you would get coffee instead. If, after pressing the t button, you were to press the c before putting in more money this action would be rejected.

This is the kind of system that we wish to model. Of course this is a very simple example but we will use it to illustrate the ideas. More sophisticated models can be built by allowing the machine to have outputs (we did not really model output with our vending machine) and by allowing different machines to *communicate*. We will not pursue this here.

2 Labelled transition systems

In our example machine there was a set of states (which the experimenter did not see) and a set of possible *actions* that we could perform on the machine. We need to make a distinction between what we see as modellers and what we see as experimenters. When we are designing the machine we, of course, see everything including the states. When we said above that "we" do not see the states we meant "we" as in the people using the machine (the experimenters).

Definition 1. A **labelled transition system** consists of:

1. A set S of states,
2. a set $\mathcal{A}$ of actions,
3. a transition relation $\longrightarrow \subseteq S \times \mathcal{A} \times S$.

The meaning of the last relation is that if $(s, a, s') \in \longrightarrow$ then starting from state s, if the system takes action a it *may* end up in state s'. I emphasize

$$\begin{tikzcd}
 & t_0 \arrow[dl, "\mathsf{m}"'] \arrow[dr, "\mathsf{m}"] & \\
t_1 \arrow[d, "\mathsf{c}"'] & & t_2 \arrow[d, "\mathsf{t}"] \\
t_3 & & t_4
\end{tikzcd}$$

Figure 2: Another vending machine LTS

the word "may" because we are allowing nondeterminism. We will write $s \xrightarrow{a} s'$ to symbolize this; the transition arrows are labelled by action names: hence the name "labelled transition system."

We will draw our systems in the same way as we drew finite automata, as illustrated in Fig. 1, except that we do not have start states nor do we have accept states. We are just interested in what actions we can see.

2.1 Observations

An important aspect of labelled transition systems is the observations. For finite automata we can see the whole word being read and we can see at the end whether it accepted or rejected the word. In a labelled transition system we see whether each action is rejected or not. Thus we can make some important distinctions that we could not make with automata.

Consider the variant vending machine shown in Fig. 2.

In this labelled transition system if we start from t_0 and perform an m action, the result is indeterminate. What possible sequences of behaviour can we see? We can see mc and mt. These are exactly the same sequences we see with the machine shown in Fig. 1. Are these two machines identical? Certainly not! In the first machine the user gets to choose between having tea and having coffee; in the second labelled transition system the machine makes the decision. A user of these vending machines will certainly notice the difference eventually. *Thus the sequences do not tell the whole story*; we need a different notion of equivalent behaviour. This notion is called bisimulation; we discuss it below.

3 A formalism for describing LTSs

Just as we had regular expressions as an algebraic language for describing regular languages which are the behaviours of finite automata, we will introduce a simple language to describe labelled transition systems. This language is an example of what is called “process algebra”; there are many versions and variations possible. The language is defined inductively.

We assume that there is a finite set of **basic actions**, which we denote by $\mathcal{A}$; this is the analogue of the alphabet used in automata theory. The base case of our inductive definition is the process NIL: as its name suggests, it does nothing. Then we have some constructs for building new processes from old ones: they are called **combinators** because they allow us to combine processes. We have two combinators (to start with): they are denoted $\cdot$ and $+$.

Definition 2. The set of **processes** is defined as follows:

1. NIL is a process,
2. if $a \in \mathcal{A}$ and P is a process, then $a \cdot P$ is a process,
3. if P_1, P_2 are processes then $P_1 + P_2$ is a process.

We also allow the use of parentheses to define grouping as in ordinary algebra.

Here are some examples of processes: NIL, $a \cdot NIL$, $a \cdot b \cdot NIL + a \cdot c \cdot NIL$ and $a \cdot (b \cdot NIL + c \cdot NIL)$. These define LTSs as follows. The NIL process does nothing so it defines an LTS with one state and no arrows. The LTSs for the other processes are shown below; I have not written names for the states.

$$\begin{tikzcd}
\bullet \arrow[d, "a"] \\
\bullet
\end{tikzcd}
\qquad
\begin{tikzcd}
& \bullet \arrow[dl, "a"'] \arrow[dr, "a"] & \\
\bullet \arrow[d, "b"] & & \bullet \arrow[d, "c"] \\
\bullet & & \bullet
\end{tikzcd}
\qquad
\begin{tikzcd}
& \bullet \arrow[d, "a"] & \\
& \bullet \arrow[dl, "b"'] \arrow[dr, "c"] & \\
\bullet & & \bullet
\end{tikzcd}$$

So process algebra terms are to LTSs as regular expressions are to DFA. However, we are interested in the behaviour of the system and not just a one-off decision at the end. If we look at the last two processes they both allow the sequences $\{ab, ac\}$ so from the point of view of ordinary automata

$$\text{ACT}\ \frac{}{a\,.\,P \xrightarrow{a} P} \qquad \text{SUM1}\ \frac{P \xrightarrow{a} P'}{P+Q \xrightarrow{a} P'} \qquad \text{SUM2}\ \frac{Q \xrightarrow{a} Q'}{P+Q \xrightarrow{a} Q'}$$

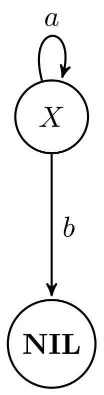
Figure 3: Operational semantics

theory they are the same; but from the point of view of behaviour they are very different. In order to make the expressions less ugly we will drop the explicit NILs and assume they are there in the right places. How do we officially describe the LTSs associated with a given process expression? We have **operational semantics** shown in Fig. 3. Each rule expresses what one must assume above the line and shows what follows below the line. From such a description we can construct an LTS by using the process expressions themselves as the names of the states.

The processes that we have defined so far do not permit any kind of looping; that is a huge limitation! In order to describe such processes we will introduce process variables and recursion. We write X for a process variable and we allow process *equations* of the form $X = P(X)$ where $P(X)$ is a process expression containing X; so this is a recursive definition of X. For example we could write

$$X = a.X + b.NIL$$

for a process. This process can do an a-action and become itself or a b-action and stop. The LTS for this is



Note that we have used the processes as the labels for the states. In general we need a rule for the operational semantics with recursion. We will write $\mathbf{rec}X.P$ to mean the process defined by the recursive definition $X = P(X)$. We will write $P[X \mapsto Q]$ to mean the expression obtained by replacing X within P by Q. So, if we write $P[X \mapsto \mathbf{rec}X.P]$ we mean: "take the expression for P (which will contain X), and replace occurrences of X by copies of the entire expression $\mathbf{rec}X.P$. Now our rule is

REC $\frac{P[X \mapsto \mathbf{rec}X.P] \xrightarrow{a} P'}{\mathbf{rec}X.P \xrightarrow{a} P'}$

With this we can express general LTSs. For example, our vending machine will be described as follows: actions are $\{\$, C, \overline{C}, T, \overline{T}\}$ and the process is given by

$$X = \$.(C.\overline{C}.X + T.\overline{T}.X)$$

or in the more formal syntax

$$\mathbf{rec}X.\$.(C.\overline{C}.X + T.\overline{T}.X).$$

4 Bisimulation

What we want our bisimulation relation to say is that each machine can match the moves of the other no matter what. We formalize the definition as follows. We will talk about different states of the same labelled transition system as being bisimilar or not rather than two systems being bisimilar or not. It is slightly more convenient and this definition can easily be adapted to the case of two different systems.

Definition 3. Given an labelled transition system $\mathcal{S} = (S, \mathcal{A}, \longrightarrow)$ we say two states s and t are **bisimilar** if for every state s' and action a such that a transition $s \xrightarrow{a} s'$ is possible there is a state t' such that the transition $t \xrightarrow{a} t'$ is possible with s' and t' being bisimilar; and, conversely if for every state t' and action a such that a transition $t \xrightarrow{a} t'$ is possible there is a state s' such that the transition $s \xrightarrow{a} s'$ is possible with s' and t' being bisimilar.

What this says is that starting from two bisimilar states the actions of one can always be matched with the actions from the other with the matching actions always chosen to go to bisimilar states. In fancier language, the bisimulation relation is a *dynamical invariant*.

If we just think of the labelled transition systems shown in the two figures above as being combined, we can ask, which states are bisimilar? The answer is that s_2, s_3, t_3, t_4 are all bisimilar; none of them can do anything, and no other pair is bisimilar.

Consider the system shown in Fig. 4.

$$\begin{tikzcd}
 & & s_0 \arrow[d, "\mathsf{a}"'] & & \\
 & & s_1 \arrow[dl, "\mathsf{b}"'] \arrow[dr, "\mathsf{b}"] & & \\
 & s_2 \arrow[dl, "\mathsf{c}"'] \arrow[dr, "\mathsf{d}"] & & s_3 \arrow[dl, "\mathsf{c}"'] \arrow[dr, "\mathsf{d}"] & \\
s_4 & & s_5 & & s_6
\end{tikzcd}$$

Figure 4: Collapsing a Labelled Transition System by bisimulation

$$\begin{tikzcd}
t_0 \arrow[d, "\mathsf{a}"'] \\
t_1 \arrow[d, "\mathsf{b}"'] \\
t_2 \arrow[d, "\mathsf{c,d}"'] \\
t_3
\end{tikzcd}$$

Figure 5: Collapsing a Labelled Transition System by bisimulation

Which states are bisimilar here? Certainly s_4, s_5, s_6 are all bisimilar since none of them can do anything. The states s_2 and s_3 are also bisimilar: if s_2 does a c-action and goes to s_4, s_3 can match it by doing a c-action and going to s_5, and s_4 and s_5 are bisimilar states. Note, the *incoming* arrows can be quite different even in bisimilar states. If we identify all the bisimilar states we get the labelled transition system shown in Fig. 5. We call this the *bisimulation-minimal* version of the transition system.

4.1 Formalizing Bisimulation

The definition of bisimulation looks as if it has a circularity present in it. It may also look like an inductive definition but with the base case missing. However, it is perfectly rigourous as it stands: it is a *co-inductive* definition rather than an inductive definition. In this subsection we will explain how to interpret it properly.

There are several ways to define bisimulation. We will present three ways to understand the definition, each of which has its advantages and disadvantages.

The first uses induction.

Definition 4. For $n \in \mathbb{N} = \{0, 1, 2, ...\}$, we define a family of equivalence relations $\sim_n$ by induction on n,

1. We say that all states s, t are **zero-equivalent**. That is,

$$\forall s, t \in S \quad s \sim_0 t$$

2. Assuming up to n-equivalence, $s \sim_{n+1} t$ holds if, for any state s' and action a for which the transition $s \xrightarrow{a} s'$ is possible, there is a state t' for which the transition $t \xrightarrow{a} t'$ is possible, and $s' \sim_n t'$; **and** if for any state t' for which the transition $t \xrightarrow{a} t'$ is possible there is a state s' for which the transition $s \xrightarrow{a} s'$ is possible and $s' \sim_n t'$

For example, in Fig. 1 and Fig. 2, states s_0, t_1, t_2 are 1-equivalent, but not 2-equivalent.

Definition 5. Bisimulation, written $\sim$, is the intersection of all i-equivalences:

$$\sim = \bigcap_n \sim_n$$

or equivalently,

$$s \sim t \text{ iff } \forall n \; s \sim_n t$$

Another, more elegant, definition of bisimulation requires the discussion of fixed points on complete lattices. Recall the notions of least upper bound and greatest lower bound in a poset. A *complete lattice* is a poset in which every set (even the empty set) has a least upper bound; it follows that every set has a greatest lower bound. The appendix reviews the definition of a lattice and a complete lattice and gives the proofs of the results that we need.

Now, consider the complete lattice of all binary relations on S where S is the state set of some labeled transtion system $(S, \mathcal{A}, \longrightarrow)$. Let $\mathcal{R}$ be this collection ordered by inclusion. $\mathcal{R}$ is a complete lattice. Define a *relation transformer* $\mathcal{F} \colon \mathcal{R} \longrightarrow \mathcal{R}$ as follows:
Definition 6. Given $R \in \mathcal{R}$ we define a relation $\mathcal{F}(R)$:
$s\,\mathcal{F}(R)\,t$ if, for every state s' for which a transition $s \xrightarrow{a} s'$ is possible, there exists a state t' for which a transtion $t \xrightarrow{a} t'$ is possible and $s'\,R\,t'$; and vice versa.

Notice the similarity between Def. 5 and Def. 4.1. Since $\mathcal{R}$ is a complete lattice and $\mathcal{F}$ defines a monotone function on $\mathcal{R}$ (this is a routine check) it has a greatest fixed point by standard results of lattice theory. The greatest fixed point of $\mathcal{F}$ is in fact $\sim$. This yields our second way of understanding the definition of bisimulation. The proof that these two ways of formalizing the definition are the same is omitted; it is not very difficult.

Finally we can formalize bisimulation as follows. I would like all of you to know this version.
Definition 7. Let S be the state space of some labelled transition system. We say that R is **a bisimulation relation** [1] on S if

$$\forall s, t \text{ s. t. } sRt \forall s' \text{ s.t. } s \xrightarrow{a} s' \exists t' \text{ s.t. } t \xrightarrow{a} t' \text{ and } s'Rt'$$

and vice versa.

What we are saying with the above definition is that R is a fixed point of $\mathcal{F}$. Note there is nothing circular about this definition. If some relation is given one can ask whether or not it satisfies the property, if it does it is a bisimulation relation. Now we say that two states are bisimilar if and only if there is some bisimulation relating them. This is closely related to the second way of formalizing bisimulation.

5 Other equivalences

Bisimulation has a nice algebraic theory and other pleasant mathematical properties but perhaps it makes too many distinctions. There are many

[1]Note the use of the indefinite article.

$$\begin{tikzcd}
 & s_0 \arrow[dl, "a"'] \arrow[d, "a"] \arrow[dr, "a"] & \\
s_1 \arrow[d, "b"'] & s_2 \arrow[dl, "b"'] \arrow[dr, "c"] & s_3 \arrow[d, "c"] \\
s_4 & & s_5
\end{tikzcd}
\qquad
\begin{tikzcd}
 & t_0 \arrow[d, "a"'] & \\
 & t_1 \arrow[dl, "b"'] \arrow[dr, "c"] & \\
t_2 & & t_3
\end{tikzcd}$$

Figure 6: Two-way simulation versus bisimulation

other equivalences that one can use to discuss systems. Which one is appropriate depends on what you want to do. Bisimulation is good for guaranteeing properties like: if A is bisimilar to B one can replace A with B in some larger system and guarantee the same overall behaviour. Here we are thinking of A as one component of a bigger system. This property is called compositionality.

At the other end of the spectrum is *trace equivalence.* This is very close to language equivalence that we have been studying with DFAs. This just says that two states are equivalent if the possible sequences of actions from the two states are the same. Thus, in our vending machine examples, the start states are trace equivalent even though they are not bisimilar.

A very important relation is *simulation.* It is not an equivalence relation; it is "one half of" of bisimulation and thus not symmetric.

Definition 8. Given a labelled transition system $\mathcal{S} = (S, \mathcal{A}, \longrightarrow)$ we say that the state t **simulates** the state s if, for every state s' and action a such that a transition $s \xrightarrow{a} s'$ is possible, there is a state t' such that the transition $t \xrightarrow{a} t'$ is possible and t' simulates s'.

This is again coinductive and can be understood as we have understood bisimulation. We define an equivalence called *two-way simulation* as follows.

Definition 9. Given an labelled transition system $\mathcal{S} = (S, \mathcal{A}, \longrightarrow)$ we say two states t and s are **two-way similar** if t simulates s and s simulates t.

One might think that this is just the same as bisimulation but this is not true! Consider the two systems shown in Fig. 6.

Here t_0 can simulate s_0 and s_0 can simulate t_0 but the two are not bisimilar. If s_0 makes an a-transition to s_1 then t_0 cannot match this by making an a-transition to a *bisimilar* state.

Let us finish this section with another way of looking at these equivalences that will help you understand what we have discussed so far. We view the definitions in terms of games. The labelled transition systems are the "boards" on which the games are played.

The Simulation Game You want to show that a state s in machine M_s can simulate a state t in machine M_t. Your opponent makes moves starting at t, following the transitions of M_t. You must match all his moves. You win when he runs out of moves. If you have a winning strategy, s simulates t.

The 2-Way Simulation Game Your opponent moves first, and may choose which state to start from. You have to start from the other state and match all his moves. You win if he ends up with no more moves. If you have a winning strategy, M_t and M_s are 2-way similar.

The Bisimulation Game Your opponent moves first, and may choose to start from any state; you have to play with the other state. You must match any move he makes. At each turn, your opponent may choose to *switch machines* and make a move from whatever state he likes in whatever machine and you must match it in your machine. You win if he runs out of moves, he wins if you fail to match his move. If you have a winning strategy, M_s and M_t are bisimilar.

One can try out these games with the two states s_0 and t_0 of Fig. 6.

6 Logical characterization of bisimulation

We define a logic, called Hennessy-Milner logic, defined inductively as follows:

$$\phi ::== \mathsf{true} | \mathsf{F} | \phi_1 \wedge \phi_2 | \neg\phi | \phi_1 \vee \phi_2 \langle a \rangle \phi | [a]\phi.$$

We write $s \models \phi$ if the formula ϕ is true in the state s. Unlike the logics that you have studied before the truth of a formula changes as you change the state. This kind of logic is called a *modal* logic. Various modal logics have proven useful in studying systems: temporal logic, Hoare logic, dynamic logic, deontic logic and epistemic logic for example. Hennessy-Milner logic is a kind of dynamic logic. These formulas are interpreted on the states of a labelled transition system as follows, we are using induction on the structure of formulas:

if	ϕ is true then	$\forall s\ \ s \models \mathsf{true}$
if	ϕ is $\phi_1 \wedge \phi_2$ then	$s \models \phi$ if and only if $s \models \phi_1$ and $s \models \phi_2$
if	ϕ is $\phi_1 \vee \phi_2$ then	$s \models \phi$ if and only if $s \models \phi_1$ or $s \models \phi_2$
if	ϕ is $\neg\psi$ then	$s \models \phi$ if and only if $s \not\models \psi$
if	ϕ is $\langle a\rangle\psi$ then	$s \models \phi$ if and only if $\exists s'$ such that $s \xrightarrow{a} s'$ and $s' \models \psi$.
if	ϕ is $[a]\psi$ then	$s \models \phi$ if and only if $\forall s'\ s \xrightarrow{a} s'$ implies $s' \models \psi$.

We need not have "or" since we can define $\phi \vee \psi = \neg(\neg\phi \wedge \neg\psi)$; so there is no need to have it except as a notational convenience. The formula $\langle a\rangle\phi$ is true in a state s if it is *possible* for s to make an a-transition to a state s' which satisfies ϕ. The formula $[a]\phi$ says that if s can make an a-transition to a state s' then s' *must* satisfy ϕ. We do not really need this formula: $[a]\phi = \neg\langle a\rangle(\neg\phi)$.

The remarkable theorem proved by Hennessy and Milner and by van Benthem is
Theorem 10. Two states are bisimilar **if and only if** they satisfy exactly the same formulas of the logic.

This gives a way of showing that two states are *not* bisimilar by finding a formula that one satisfies and the other does not.

Let us consider some examples. Consider the system shown below

$$\begin{tikzcd}
s_1 \arrow[r, "a"] \arrow[d, "c"'] \arrow[dr, "a"] & s_2 \arrow[r, "b"] & s_3 \\
s_4 \arrow[d, "b"'] \arrow[dr, "b"] & s_5 \arrow[ur, "b"] & \\
s_6 & s_7 \arrow[u, "c"'] &
\end{tikzcd}$$

In this, the state s_3 does not satisfy any formula of the form $\langle a\rangle\phi$ because it cannot do an a-transition. It satisfies *any* formula of the form $[a]\phi$ precisely because it cannot do an a-transition from s_3. We have also $s_1 \models [a]\langle b\rangle\mathsf{T}$, $s_4 \models \langle b\rangle\langle c\rangle\mathsf{T}$.

Our crucial examples are the vending machine shown in Figs. 1 and 2 and reproduced below (with labels changed for simplicity).

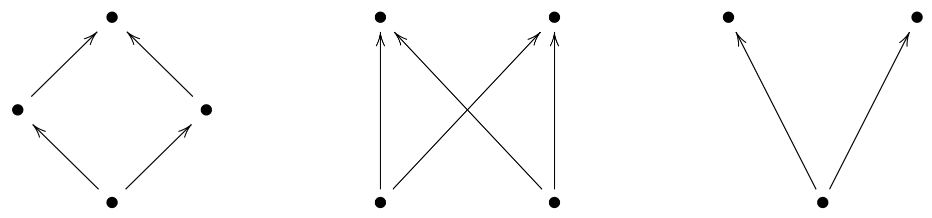
$$\begin{tikzcd}
 & s_0 \arrow[d, "a"'] & & & t_0 \arrow[dl, "a"] \arrow[dr, "a"'] & \\
 & s_1 \arrow[dl, "b"'] \arrow[dr, "c"] & & t_1 \arrow[d, "b"] & & t_2 \arrow[d, "c"'] \\
s_2 & & s_3 & {} & t_3 & t_4
\end{tikzcd}$$

Now we know that s_0 and t_0 are not bisimilar, on what formulas do they disagree? Here we have many choices. Consider the formula $\langle a\rangle(\neg\langle b\rangle\mathsf{T})$, this is satisfied by t_0 but not by s_0. We can also write this as $\langle a\rangle[b]\mathsf{F}$. Note that $[a]\phi \equiv \neg\langle a\rangle\neg\phi$ so a boxed formula is implicitly negative. What can we do if we only allow positive formulas? It turns out that positive formulas are not enough to distinguish all non-bisimilar systems.

A Fixed points on complete lattices

Definition 11. A **lattice** is a partially ordered set in which every pair of elements has both a least upper bound and greatest lower bound.

For example, only the first of these three is a lattice:



For another example, consider the power set of any set X ordered by inclusion. Its supremum is the union of all subsets; its infimum, the intersection of all subsets.

A lattice does not necessarily need a top or a bottom. For example, the rational numbers form a lattice with no top or bottom.

Definition 12. A **complete lattice** is a lattice in which every subset (even $\emptyset$) has a least upper bound.

Proposition 13. In a complete lattice, every subset also has a greatest lower bound.

Proof:
Let L be a complete lattice and $X \subset L$. Consider $X\downarrow := \{u : \forall x \in X.u \leq x\}$, and let w be the least upper bound of $X\downarrow$.

Claim: w is the greatest lower bound of X

For all $x \in X$, x is an upper bound of $X\downarrow$. Now w is the least upper bound of $X\downarrow$, so for all $x \in X$, $w \leq x$. So w is a lower bound of X. So $w \in X\downarrow$, and since it is an upper bound of $X\downarrow$, w is the greatest lower bound of X.

Definition 14. A function $f : X \to X$ is **monotone** if $\forall x \in X$

$$x \leq y \quad \Rightarrow \quad f(x) \leq f(y)$$

Definition 15. A point x is a **fixed point** of a function f if

$$f(x) = x$$

Theorem 16. Every monotone function $f : L \to L$ on a complete lattice L has both a least fixed point and a greatest fixed point.

Proof:
Consider all the points which are (weakly) decreased by f

$$M := \{x \in L \,|\, x \leq f(x)\} \qquad v := \text{least upper bound of } M$$

Claim: v is a fixed point. To show this, note that since v is an upper bound of M, for any $x \in M$ $x \leq v$ so, since f is monotone, $f(x) \leq f(v)$. Combining with $x \leq f(x)$ using transitivity, we have $x \leq f(v)$.
Thus, $f(v)$ is an upper bound of M,
so $v \leq f(v)$
so, by monotonicity of f, $f(v) \leq f(f(v))$,
so $f(v) \in M$
so $f(v) \leq v$
so $v = f(v)$, i.e. v is a fixed point of f
Claim: v is the greatest fixed point of f
Any fixed point is in M, and v is an upper bound of M. So v is the greatest fixed point of f.

The other part of the theorem is proved similarly for

$$M' := \{x \in L \,|\, f(x) \leq x\} \qquad v := \text{greatest lower bound of } M$$