This last chapter deals with the use of temporal logic in the verification of distributed systems. Temporal logic is appropriate to represent and to verify liveness properties. The other important feature is that with the help of temporal logic it is possible to verify single properties; it is not necessary to model services. Modelling of services is often difficult.

There are many kind of temporal logics. The most fundamental division divides the logics into two classes: linear time and branching time logics.
Traditionally, temporal logics are not based on labelled transition systems but on *Kripke structures*. This enforces us to introduce Kripke structures in this course. These structures resemble transition systems, but transitions are without labels. Instead, states contain information which can be analysed with the help of temporal logic. There is, however, a complete correspondence between transition systems and Kripke structures.

We introduce Kripke structures, their correspondence with transition systems, linear time logic LTL and the basics of branching time logic CTL.

There are similar systems which have been developed for transition systems. ACTL is a straightforward modification of CTL for transition systems. One other important formalism is $\mu$-calculus which is also near CTL. We do not deal with these modifications in this course.
Traditionally, temporal logic uses models where processes are represented as graphs in such a way that states contain information. Transitions are without labels. In order to handle temporal logics in a formal way, we must somehow define the data in the states. This is done as follows.
Definition

Kripke structure is a tuple $\mathcal{K} = (S, AP, L, \rightarrow, S_0)$, where

- $S$ is a set of states;
- $AP$ is a finite non-empty set of atomic propositions;
- $L : S \rightarrow \mathcal{P}(AP)$ is a function that labels each state with the set of atomic propositions true in that state;
- $\rightarrow \subseteq S \times S$ is a transition relation that must be total, that is, for every state $s$ there is a state $s'$ such that $(s, s') \in \rightarrow$; the element $(s, s') \in \rightarrow$ is called a transition and it is denoted $s \rightarrow s'$;
- $S_0$ is the set of initial states.
Example. Consider the following transition system (Dekker’s mutual exclusion algorithm)
Next we show the same as a Kripke structure. As a graph, the Kripke structure is the same as the transition system, see figure.
We must add the propositions. Let us take two atomic propositions, $p_1$ and $p_2$. Proposition $p_1$ says that process 1 is at the critical area and $p_2$ says that process 2 is at the critical area.

In the original (transition) graph $p_1$ is true in states 4, 6 and 7. Similarly, $p_2$ is true in states 3, 5 and 11. In other states $p_1$ and $p_2$ are false.

These conditions determine the set $AP$ and the function $L$. The mutual exclusion can now be expressed as a sentence “Both $p_1$ and $p_2$ are not true at the same time at any state”.
In the above example the Kripke structure was formed ad hoc. It can be shown that the transformation is always possible.

**Proposition**

*Every labelled transition system can be transformed into an equivalent Kripke structure.*

**Proof.**

- Let $\textit{LTS} = (S, A, T, s_0)$ be a labelled transition system. Let us construct a Kripke structure $\mathcal{K} = (S', AP, L, \rightarrow, 0_S)$ as follows.

- The set of states $S' = S \times A$. Between states $(s_1, a_1)$ and $(s_2, a_2)$ there is a transition in $\mathcal{K}$, $(s_1, a_1) \rightarrow (s_2, a_2)$, if and only if there is a state $s \in S$ such that

$$s_1 \xrightarrow{a_1} s_2 \xrightarrow{a_2} s$$

in $\textit{LTS}$. This defines a transition relation $\rightarrow$ in $\mathcal{K}$. 


The set of initial states in $\mathcal{K}$ is

$$S_0 = \{(s_0, a_0) \in S' \mid s_0 \xrightarrow{a_0} \text{ in } \text{LTS}\}.$$

Let $\eta$ be a variable that can get an action as its value. The propositions are of the form $\eta = a$, where $a \in A$.

The interpretation of the proposition is that if $\eta = a$ is true in some state in $\mathcal{K}$, the system can next perform $a$ in that state. Set now $L : S' \to \mathcal{P}(AP)$ by defining $(s, a) \mapsto (\eta = a)$. □

Before we formed the Kripke structure of Decker’s algorithm manually ad hoc. If the same is done with the help of the theorem’s construction, we will get the following structure (of which only half is drawn).
The vice versa transformation is possible, too.

**Proposition**

*Every Kripke structure can be transformed into an equivalent transition system.*

**Proof.** Exercise. Hint: Labels must be sets of propositions. □
Linear time logic was the first temporal logic that was applied in the verification of distributed computer systems. Its main developers were Manna and Pnueli.

The starting point of LTL is two operators, $X$ and $U$. If $p$ is a proposition, then $Xp$ means that next time or in the next state $p$ is true.

The formula $pUq$, on the other hand, claims that $p$ is true until $q$ is true. We define the formulas of LTL formally as follows:
Definition

Let $\text{AP}$ the set of atomic propositions. A LTL formula is defined inductively:

1. If $\phi \in \text{AP} \cup \{\top, \bot\}$, then $\phi$ is a formula.
2. If $\phi$ and $\psi$ are formulas, then also $(\neg \phi)$, $(\phi \land \psi)$, $(\phi \lor \psi)$, $(\phi \rightarrow \psi)$ and $(\phi \equiv \psi)$ are formulas.
3. If $\phi$ and $\psi$ are formulas, then also $X\phi$ and $\phi U \psi$ are.
Linear Time Logic LTL III

- The semantics of the formulas are defined formally with the help of Kripke structures and their paths. A path $\pi$ in a Kripke structure $\mathcal{K}$ is a finite or infinite sequence of states, $s_0s_1\cdots s_n$ or $s_0s_1\cdots$.

- State $s_0$ is always the initial state of a Kripke structure and there is a transition between successive states. A path can be finite only, if it ends at a state where there are no out-going transitions.

- Sometimes it is demanded that all the states must have out-going transitions. Then only infinite paths are possible. There is no difference between these two agreements in practice.

- If $\pi$ is a path $s_0s_1\cdots$, then $\pi^k$ means the part $s_k s_{k+1} \cdots$ of the path.

- The truth of a LTL formula is defined first with respect to a path $\pi$. If $\phi$ is a formula and it is true with respect to $\pi$, we denote $\pi \models \phi$.

- In what follows we define the truth of a formula precisely. The truth of the constants $\top$ and $\bot$ is defined in such a way that the former is true in every state and the latter is false in every state.
Definition

The truth of a formula $\phi$ with respect to a path $\pi = s_0s_1 \cdots$ is defined as follows:

- If $\phi \in A \cup \{\bot, \top\}$ is an atomic proposition or a constant, then $\pi \models \phi$ iff $s_0 \models \phi$ (i.e. $\phi \in L(s_0)$ or $\phi$ is $\top$).
- $\pi \models \phi_1 \lor \phi_2$ iff $\pi \models \phi_1$ or $\pi \models \phi_2$.
- $\pi \models \phi_1 \land \phi_2$ iff $\phi \models \phi_1$ and $\pi \models \phi_2$.
- $\pi \models \neg \phi$ iff $\pi \not\models \phi$.
- $\pi \models X\phi$ iff $\pi^1$ exists and $\pi^1 \models \phi$.
- $\pi \models \phi_1 U \phi_2$ iff $\pi \models \phi_2$ or there exists $k > 0$ such that $\pi^k$ is defined, $\pi^k \models \phi_2$ and for all $i$, $0 \leq i < k$, $\pi^i \models \phi_1$. 
State $s$ satisfies formula $\phi$ in a Kripke structure $\mathcal{K}$, $\mathcal{K}, s \models \phi$, iff for every path $\pi$, starting from $s$, $\pi \models \phi$.

Formula $\phi$ is true in $\mathcal{K}$, $\mathcal{K} \models \phi$, iff $\pi \models \phi$ for every path $\pi$ in $\mathcal{K}$. 
Consider the formula $\top U \phi$. Because $\top$ is true in every state, $\top U \phi$ is true on the path $\pi$ iff $\pi^k \models \phi$ for some $k \geq 0$. Thus $\top U \phi$ is true, if $\phi$ is true at some point in the future. We use the notation $F \phi$ for this formula.

Formula $G \phi$ says that $\phi$ is true at every state on the path. It could be expressed with the help of $F$ by writing $\neg F \neg \phi$.

Formula $\phi R \psi$ says that after a finite amount of steps $\phi$ is true, and before that $\psi$ is true in every state including the first state where $\phi$ is true.

There are alternative symbols for the operators we have just defined:

- $X = \bigcirc$,
- $G = \Box$,
- $F = \Diamond$,
- $U = \mathcal{U}$.
These symbols are traditional, but the newer symbols are easier to remember. We use both. Next we present some general laws using the traditional symbols:

- Operators □ and ◊ are dual:

\[ \neg □ \Phi \equiv ◊ \neg \Phi. \]

- ◊ can be written with the help of \( \mathcal{U} \) (as we have shown before):

\[ ◊ \Phi \equiv \top \mathcal{U} \Phi. \]

- ◊ is distributive with respect to \( \vee \) and □ with respect to \( \wedge \):

\[ ◊ (\Phi \vee \Psi) \equiv ◊ \Phi \vee ◊ \Psi, \]
\[ □ (\Phi \wedge \Psi) \equiv □ \Phi \wedge □ \Psi. \]
In addition,

\[ \neg \Box \Phi \equiv \Box \neg \Phi, \]
\[ \neg (\Phi U \psi) \equiv (\neg \psi U (\neg \Phi \land \neg \psi)) \lor \square \neg \psi. \]
Next we present typical situations where LTL expressions are easy to use:

1. Mutual exclusion:
   \[ G \neg (\text{critical}_1 \land \text{critical}_2) \]

2. At most one request is acknowledged:
   \[ \bigwedge_{i<j} G \neg (\text{ack}_i \land \text{ack}_j) \]

3. Liveness property, according to which my turn is infinitely often:
   \[ GF \text{ myturn} \]

4. Liveness property that if succeeding to enter to try-area leads finally to the critical area:
   \[ G(\text{try} \rightarrow F\text{critical}) \]
After initialization the system stays initialized:

\[ FG \text{ initialized} \]

However, it is not easy to express all the useful properties in LTL. Take for example the following property of an elevator:

Between the times the elevator is asked to a certain floor and it opens its door at that floor, the elevator can pass the floor at most two times:

\[
G( (\text{call} \land F\text{open}) \rightarrow
\left((\neg\text{atfloor} \land \neg\text{open})U
\left(\text{open} \lor \left((\text{atfloor} \land \text{open})U
\left(\text{open} \lor \left((\neg\text{atfloor} \land \neg\text{open})U
\left(\text{open} \lor \left((\text{atfloor} \land \neg\text{open})U
\left(\text{open} \lor \left((\neg\text{atfloor} \land \text{open})))))))))))))\)
\]
Consider the Kripke structure of another mutual exclusion algorithm:

N=noncritical, T=trying, C=critical

User1, User2

N1, N2
turn=0

T1, N2
turn=1

C1, N2
turn=1

C1, T2
turn=1

T1, T2
turn=1

T1, T2
turn=1

T1, T2
turn=2

T1, C2
turn=2

N1, T2
turn=2

N1, C2
turn=2

N1, T2
turn=2

N1, C2
turn=2

T1, C2
turn=2
Example II

Let us examine how various formulas behave with respect to this structure.

- **Formula**
  \[ \Box \neg (C_1 \land C_2) \]
  is true, because the processes are never at the critical area at the same time.

- **On the other hand, formula**
  \[ \Diamond C_1 \]
  is not true, because there is an infinite path from the initial state such that User1 has not tried to enter the critical area.

- **But formula**
  \[ \Box (T_1 \Rightarrow \Diamond C_1) \]
  is true. It means that if process User1 tries to enter the critical area, it succeeds to enter the area in the future. Thus the algorithm is fair.
Example III

- Formula
  \[ \Box \Diamond C_1 \]
  is not true, because without trying a process can never enter the critical area.

- Furthermore,
  \[ \Box \Diamond T_1 \Rightarrow \Box \Diamond C_1 \]
  is true, because at every path where \( T_1 \) is performed infinitely many times, the critical area is also entered infinitely many times. Thus the algorithm is fair in a strong sense.
Another formalism is CTL *Computational Tree Logic*. It adds four new operators to the selection of operators

- $EX$ (existential),
- $AX$ (all),
- $E(\cdot U \cdot)$,
- $A(\cdot U \cdot)$.

**Definition**

A formula in CTL is defined inductively:

1. If $\phi \in AP \cup \{\top, \bot\}$, then $\phi$ is a formula.
2. If $\phi$ and $\psi$ are formulas, then also ($\neg \phi$), ($\phi \land \psi$), ($\phi \lor \psi$), ($\phi \to \psi$) and ($\phi \leftrightarrow \psi$) are formulas.
3. If $\phi$ and $\psi$ are formulas, then $EX\phi$, $AX\phi$, $E(\phi U \psi)$ and $A(\phi U \psi)$ are, too.
The truth of a formula is defined with respect to a Kripke structure $M$ and its state as follows:

- If $\phi \in AP \cup \{\bot, \top\}$, then $M, s_0 \models \phi$ iff $s_0 \models \phi$.
- $M, s_0 \models \phi \lor \psi$, iff $M, s_0 \models \phi$ or $M, s_0 \models \psi$.
- $M, s_0 \models \phi \land \psi$, iff $M, s_0 \models \phi$ and $M, s_0 \models \psi$.
- $M, s_0 \models \neg \phi$, iff $M, s_0 \not\models \phi$.
- $M, s_0 \models \text{EX} \phi$, iff there exists $s' \in S$ such that $s_0 \rightarrow s'$ ja $M, s' \models \phi$.
- $M, s_0 \models \text{AX} \phi$, iff for all $s' \in S$: if $s_0 \rightarrow s'$, then $M, s' \models \phi$.
- $M, s_0 \models \text{E}(\phi U \psi)$, iff
  - $M, s_0 \models \psi$ or
  - there is a path $\pi = s_0 \rightarrow s_1 \rightarrow \cdots \rightarrow s_n \rightarrow \cdots$, such that there is $k > 0$ and $M, s_k \models \psi$ and for all $i$, $0 \leq i < k$, $M, s_i \models \phi$.
- $M, s_0 \models \text{A}(\phi U \psi)$, iff
  - $M, s_0 \models \psi$ or
for all infinite paths $\pi = s_0 \rightarrow s_1 \rightarrow \cdots \rightarrow s_n \rightarrow \cdots$ there is $k > 0$ such that $M, s_K \models \psi$ and for all $i$, $0 \leq i < k$, $M, s_i \models \phi$; or

for all finite paths $\pi = s_0 \rightarrow s_1 \rightarrow \cdots \rightarrow s_n$, $M, s_i \models \phi$ for all $i$, $0 \leq i \leq n$.

Usually the following operators are used in CTL:

1. $AX$: for all paths next is true.
2. $EX$: there is a path such that next is true.
3. $AF$: for all paths in the future.
4. $EF$: for some path in the future.
5. $AG$: for all paths always.
6. $EG$: for some path always.
7. $AU$ or $A[f \cup g]$: for all paths $f$ until $g$.
8. $EU$ or $E[f \cup g]$: for some path $f$ until $g$. 
AR or $A(fRg)$: for all paths $g$ until $f$ including the state where $f$ is true for the first time. However, it can happen that $f$ is false in every state.

ER or $E(fRg)$: for some path $g$ until $f$ including the state where $f$ is true for the first time. However, $f$ can be false in every state.
In the following diagram we see the interpretation of the four most usual operators.

\[ M, s_0 \models \text{EF} \, g \]

\[ M, s_0 \models \text{AF} \, g \]

\[ M, s_0 \models \text{EG} \, g \]

\[ M, s_0 \models \text{AG} \, g \]
Typical CTL formulas are:

1. \( EF(Start \land \neg Ready) \): It is possible to reach a state where \( Start \) is true but \( Ready \) is false.
2. \( AG(Req \rightarrow AF Ack) \): If a request is done, it will be acknowledged.
3. \( AG(AF DeviceEnabled) \): Proposition \( DeviceEnabled \) is true infinitely often on every path.
4. \( AG(EF Restart) \): It is possible to reach \( Restart \) from every state.

LTL is not contained in CTL and CTL is not contained in LTL. For example, LTL cannot express that there exists a path, but CTL can, \( EX \phi \). On the other hand, CTL cannot express liveness properties such as LTL formula

\[
GF(\nu = \text{tick}) \rightarrow GF(\nu = \beta).
\]

That is why there is CTL* which contains both LTL and CTL. But we do not consider it during this course.
Model checking means a technique with which one constructs a finite model and tests that a certain property is true in that model. As for the temporal logic, this means that a property is expressed using temporal logic, a Kripke structure is generated from the system and then it is tested, if the logical formula is true in that structure or not.

Deciding the truth value of a formula differs essentially in LTL and CTL. In LTL, it is more difficult, because the best algorithms are PSPACE complete, although in many practical situations it is easier. Instead, computing the truth value of a CTL formula is polynomial.

That is why we start our algorithmic studies with CTL. The reference is the book Clarke, Grumberg, Peled, *Model Checking*.

- Let $\mathcal{K} = (S, R, L, O_{s_0})$ a Kripke structure and $F$ CTL formula.
The algorithm will work in such a way that it attaches information about the truth values of subformulas to every state. Let us denote the set of subformulas in the state $s$ by $\text{label}(s)$.

Initially, $\text{label}(s)$ is $L(s)$. After this the algorithm starts a loop in such a way that during the $i$th loop all the subformulas of $F$ containing $i-1$ nested operators will get a truth value. At the end $M, s \models F$ iff $F \in \text{label}(s)$.

Any formula can be expressed with the help of the operators $\neg$, $\lor$, $\text{EX}$, $\text{EU}$ and $\text{EG}$.

Thus a model checking algorithm needs to consider only six cases: $f$, $\neg f_1$, $f_1 \lor f_2$, $\text{EX} f_1$, $E[f_1 U f_2]$ ja $\text{EG} f_1$, where $f$ is atomic and $f_1$ and $f_2$ are constructed using the previous five operators.

A prerequisite is that an arbitrary CTL formula has been transformed into one of those six cases. This can be done algorithmically, but we do not consider this algorithm.
If a formula is of the form \( \neg f \), mark those states where \( f \) is false.

If a formula is of the form \( f_1 \lor f_2 \), mark those states where either \( f_1 \) or \( f_2 \) is true.

If a formula is of the form \( \text{EX } f \), mark those states whose some predecessor is marked with \( f \).

Formula \( g = E[f_1 U f_2] \) demands more work. First search all those states where \( f_2 \) is true. Then go backwards along the arcs from the found states and check, if \( f_1 \) is true in all the states on the path from the initial state to the \( f_2 \)-marked state. If it is, mark \( g \) also true in the same states. Continue until a state is reached where \( f_1 \) is not true.
procedure CheckEU($f_1, f_2$)

$T := \{s \mid f_2 \in \text{label}(s)\}$;

for all $s \in T$ do \text{label}(s) := \text{label}(s) \cup \{E[f_1 U f_2]\}$;

while $T \neq \emptyset$ do

choose $s \in T$;

$T := T \setminus \{s\}$;

for all $t$ such that $R(t, s)$ do

if $E[f_1 U f_2] \notin \text{label}(t)$ and $f_1 \in \text{label}(t)$ then

$\text{label}(t) := \text{label}(t) \cup \{E[f_1 U f_2]\}$;

$T := T \cup \{t\}$;

end if;

end for all;

end while;

end procedure;
The case $g = EG \, f$ is more complicated. Its solution is based on strongly connected components. Let $\mathcal{K}$ be a Kripke structure. We construct $\mathcal{K}'$ from $\mathcal{K}$ by leaving out all the states where $f$ is false. Of course, we delete also all the arcs and true propositions connected to these states.

**Proposition**

$\mathcal{K}, \ s \models EG \, f$, iff the following two conditions are satisfied:

1. $s \in S'$.

2. *There is path in $\mathcal{K}'$, such that it leads from $s$ to a state $t$ in some non-trivial strongly connected component (component is non-trivial if it contains more than one state).*

**Proof.**

- Suppose that $\mathcal{K}, \ s \models EG \, f_1$. Clearly $s \in S'$.

- Let $\pi$ be an infinite path which starts from $s$ and in whose every state $f$ is true.
Because $\mathcal{K}$ is finite, we can write $\pi$ in the form $\pi = \pi_0\pi_1$, where $\pi_0$ is a finite prefix of $\pi$ and $\pi_1$ is an infinite suffix of $\pi$ with the property that each state on $\pi_1$ occurs infinitely often. Then $\pi_0$ is contained in $S'$.

Let $C$ be the set of states in $\pi_1$. Clearly, $C$ is contained in $S'$.

We now show that there is a path within $C$ between any pair of states in $C$. Let $s_1$ and $s_2$ be states in $C$. Pick some instance of $s_1$ on $\pi_1$. By the way $\pi_1$ was selected, we know that there is an instance of $s_2$ further along $\pi_1$. The segment from $s_1$ to $s_2$ lies entirely within $C$. This segment is a finite path from $s_1$ to $s_2$ in $C$.

Thus, either $C$ is a strongly connected component or it is contained within one. In either case, both conditions (1) and (2) are satisfied.

Next, assume that Conditions (1) and (2) are satisfied. Let $\pi_0$ be a path from $s$ to $t$.

Let $\pi_1$ be a finite path of length at least 1 that leads from $t$ back to $t$. The existence of $\pi_1$ is guaranteed because $t$ is a state in a non-trivial strongly connected component.
All the states on the infinite path $\pi = \pi_0\pi_1$ satisfy $f_1$. Since $\pi$ is also a possible path starting at $s$ in $M$, we see that $K, s \models EG f_1$. 

□
procedure CheckEG(f₁)
  \( S' := \{ s \mid f₁ \in \text{label}(s) \} \);
  SCC := \{ C \mid C \text{ is a nontrivial SCC of } S' \} ;
  T := \bigcup_{C \in \text{SCC}} \{ s \mid s \in C \} ;
  \text{for all } s \in T \text{ do } \text{label}(s) := \text{label}(s) \cup \{ \text{EG } f₁ \} ;
  \text{while } T \neq \emptyset \text{ do }
    \text{choose } s \in T ;
    T := T \setminus \{ s \} ;
  \text{for all } t \text{ such that } t \in S' \text{ and } R(t, s) \text{ do }
    \text{if } \text{EG } f₁ \not\in \text{label}(t) \text{ then }
      \text{label}(t) := \text{label}(t) \cup \{ \text{EG } f₁ \} ;
      T := T \cup \{ t \} ;
      \text{end if;}
  \text{end for all;}
  \text{end while;}
end procedure;
Example: Microwave oven I

1. Start
2. ~Close
3. ~Heat
4. ~Error
5. Start
6. Close
7. ~Heat
8. ~Error

- Open door
- Close door
- Reset
- Start oven
- Cook
- Warmup
- Done
- Start cooking
Example: Microwave oven II

Let the formula to be examined be

$$AG(\text{Start} \rightarrow \text{AF Heat}).$$

The formula claims that in every run starting of the microwave oven leads to the situation where the oven is hot. We can write the formula in the form

$$\neg EF(\text{Start} \land \text{EG} \neg \text{Heat}).$$

First, let us determine the states where either \textbf{Start} or \textbf{\neg Heat} is true:

\begin{align*}
\text{Start} & : \{2, 5, 6, 7\} \\
\text{\neg Heat} & : \{1, 2, 3, 5, 6\}
\end{align*}
Example: Microwave oven III

Next we determine the strongly connected components, where $\neg \text{Heat}$ is true. We get one component $\{1, 2, 3, 5\}$. Every state of the component is marked with the formula $EG \neg \text{Heat}$. On the other hand, the while loop in the algorithm does not bring new states to be marked.

Recall that $EF f = E[\text{true } U f]$. When we mark with $S(f)$ those states where $f$ is true, we get

$$S(EF(\text{Start} \neg \text{Heat})) = S(E[\text{true } U (\text{Start } \land EG \neg \text{Heat})]).$$

Thus the calculation of the states satisfying $S(EF(\text{Start} \neg \text{Heat}))$ is started by defining the states where $\text{Start } \land EG \neg \text{Heat}$ is true. We get

$$T = \{2, 5\}.$$

After this, we traverse from 2 and 5 backwards and mark every found state with

$$EF(\text{Start} \neg \text{Heat})$$
Example: Microwave oven IV

. We get the states

\[ 1, 2, 3, 4, 5, 6, 7. \]

The negation of the formula is done by taking a complement with respect to all the states. Thus

\[ S(\neg EF(\text{Start} \neg \text{Heat})) = \emptyset. \]

Because the initial state of the Kripke structure does not belong to this set, the structure does not satisfy the formula.

The reason for this can be found by examining special ways to use the oven. If the oven is started and after this only door is opened and closed continually, warming does not start. In the same way, if the oven is started and door is closed, but the oven is reset just after this, warming does not start.
These kind of execution paths are connected to the concept of fairness. The previous way to use the oven is not fair.

Fairness can be taken into account in a simple way. It is enough to define a state set $F$ and demand that in a fair and infinitely long executions these states appear infinitely often. After this, it is possible to check only paths satisfying this requirement.

As for the micro wave, we can choose $F$ to be the set of the states Start, Close and Error. On these paths, the model checking formula is valid.