The global state graph can be generated mechanically from the descriptions of the processes in the system. If the graph is small, let us say less than one million states, it is possible to examine the graph systematically and search for mistakes, for example deadlocks, livelocks (it is not possible to go back to main cycle) etc.

It is difficult, however, to find all the mistakes in this way. Messages may disappear without deadlocks, the same message may be delivered two times to a receiver etc.

If a graph is too large or even infinite, it is still possible to make a random walk in the graph. If there is a mistake in a specification, it turns out often in many places in the global state graph. Even if we examine only 5 percent of the nodes, practical experiments have shown that most of the mistakes are revealed.
If we want to verify the specification more completely, we need a different approach. There are two basic approaches: equivalences and temporal logics. We consider first methods based on equivalences.

The central concept in the methods based on equivalences is a *service description*. This is a transition system which describes the service the protocol gives to its user (environment, observer). Let us examine two examples.

The AB-protocol offers a data transfer service. The protocol takes data packets from the environment (upper layer) and delivers them to a receiving participant (again upper layer in another computer). Thus it is easy to describe the service of the AB-protocol:
If we base our verification on equivalences, the next step is compare the global state graph of the AB-protocol to the global state graph of the service. If they are same in some respect (specified later in more detail), the AB-protocol can be considered correct. It seems clear that we must abstract away many details from the graph of the AB-protocol. How this is done depends on the equivalence.

Our second example is the client/server system shown in the chapter 2, when there are four clients.
Modelling Services IV

\[ c_1 \xrightarrow{t(i+1)} c_4 \xrightarrow{bci} c_3 \xrightarrow{t(i+1)} c_5 \xrightarrow{bci} c_2 \xrightarrow{cs_i} c_3 \]
Modelling Services VI

![Diagram showing transitions between B1 and B2 with labels sbi and bci]
The whole system described with the help of the parallel operator:

\[
\text{SystemRR := System} \ |
\]

Server

\[
\mid [cs1, sb1, cs2, sb2, cs3, sb3, cs4, sb4] \mid
\]

((Client1 \ |[bc1]| Buffer1 )

\[
\mid [t1, t2] \mid
\]

((Client2 \ |[bc2]| Buffer2 )

\[
\mid [t3] \mid
\]

((Client3 \ |[bc3]| Buffer3 )

\[
\mid [t4] \mid
\]

(Client4 \ |[bc4]| Buffer4 )))}
Suppose now that we are interested in the round robin principle. The principle is realized in the system, if the tokens $t_i$ travels in the following order:

$$t_2, t_3, t_4, t_1.$$ 

Thus the round robin principle is described by the following process.
We could now generate the global state graph of the process SystemRR and examine, if the token travels in that order. If we do not want to examine the graph manually, we should write a program that does the task. This would be time-consuming.

A quicker method is to compare the process SystemRR to the process RR.

The round robin principle is realized in the system, if the functioning of SystemRR, abstracted in a suitable way, corresponds the functioning of RR.
In this case, a suitable abstraction is such that all the actions except $t_1$, $t_2$, $t_3$ ja $t_4$ are changed to invisible action $\tau$. If after this change we traverse the paths in the global state graph of SystemRR, then the actions $t_i$ should appear in the RR-order and all the other actions are invisible.

A strong side of the process algebras is that it is possible to define several equivalences for different purposes. Moreover, these equivalences can be effectively computed, at least in most cases.

The simplest equivalence is *the trace equivalence*. One of the most used equivalence is *the weak bisimulation equivalence* which is sufficient for the most purposes. It can be computed effectively and it is included nearly in all the verification programs. We consider only these two in this course.
The Definition of a Relation I

Let $A$ and $B$ be sets. Every set $R \subseteq A \times B$ is called a relation from the set $A$ to the set $B$. The set

$$M_R = \{x \in A \mid \exists y \in B \text{ such that } (x, y) \in R\}$$

is the domain of $R$ and the set

$$A_R = \{y \in B \mid \exists x \in A \text{ such that } (x, y) \in R\}$$

is its range or codomain. If $R \subseteq A \times A$, i.e. if $R$ is a relation from $A$ to $A$, then it is said that $R$ is a relation in $A$. If $R$ a relation in $A$ and if $(x, y) \in R$, then we usually use the notation $xRy$. 
We define first some concepts that are useful when defining equivalence relations.

Sets $A$ and $B$ are disjoint, if $A \cap B = \emptyset$. If $\mathcal{I}$ is a set, whose elements are sets, then $\mathcal{I}$ is disjoint, if any two elements in $\mathcal{I}$ are always disjoint.

If $\mathcal{H}$ is a collection of subsets of $X$, then it is a partition of $X$, if it satisfies the following conditions:
H1. Every $A \in \mathcal{H}$ is non-empty,
H2. the union of the sets in $\mathcal{H}$ is $X$,
H3. $\mathcal{H}$ is disjoint.
The concept of a partition is closely connected to the equivalence relations which are defined next.
Definition

The relation $R$ in $X$ is an equivalence, if it satisfies the following conditions:

E1. $aRa$ for every $a \in X$ (reflexive);
E2. if $aRb$, then $bRa$ (symmetric);
E3. if $aRb$ and $bRc$, then $aRc$ (transitive).

If $a \in X$, then the set $R(a) = \{x \in X \mid aRx\}$ is the equivalence class of $a$ with respect to the equivalence $R$.

Theorem. If $R$ is an equivalence in $X$, then the set $X/R$ of the equivalence classes is a partition in $X$. For elements $a$ and $b$ in $X$, $aRb$ if and only if $a$ and $b$ belong to the same equivalence class.
Proof. Let us go through the proof, although the same is done in the elementary courses in mathematics. Let \( a \in X \). Because \( aRa \), then \( a \in R(a) \). It follows that every equivalence class is non-empty and that the union of the classes is \( X \). In order to prove condition H3 it is enough to show that given two equivalence classes, they are either identical or disjoint. Suppose that \( R(a) \cap R(b) \neq \emptyset \). Then there exits an element \( c \in R(a) \cap R(b) \). Let \( x \in R(a) \), i.e. \( aRx \). Because \( c \in R(a) \), it follows \( aRc \) and also \( cRa \), because every equivalence relation is symmetric. Because \( cRa \) and \( aRx \), then \( cRx \), because of transitivity. Because \( c \in R(b) \), also \( bRc \). This shows that \( R(a) \subseteq R(b) \). Exactly in the same way it is shown that \( R(b) \subseteq R(a) \). Thus \( R(a) = R(b) \) and the union of the equivalence classes is a partition of \( X \).

Let \( aRb \). Then \( b \in R(a) \), and thus \( a \) and \( b \) belong to the same equivalence class \( R(a) \). Conversely, suppose that \( a \) ja \( b \) belong to the same class \( R(c) \). Then \( a \in R(c) \cap R(a) \) and hence \( R(c) \cap R(a) \neq \emptyset \). Now the first part of
the proof shows that $R(c) = R(a)$. It follows $b \in R(c) = R(a)$ or $aRb$.
Thus the latter claim is true, too □

**Theorem.** Let $\mathcal{H}$ be a partition of $X$. If we define for elements $a$ and $b$ of $X$ that $aR_{\mathcal{H}}b$ if and only if $a$ and $b$ belong to the same set $U \in \mathcal{H}$, then $R_{\mathcal{H}}$ is an equivalence relation of $X$ such that the union of all the $R_{\mathcal{H}}$-equivalence sets is $\mathcal{H}$.

**Proof.** Let $a \in X$. Because $\mathcal{H}$ is a partition of $X$, there exists a set $U \in \mathcal{H}$ such that $a \in U$. Now $a$ and $a$ belong to the same set $U$ so that $aR_{\mathcal{H}}a$.
Thus the relation $R_{\mathcal{H}}$ is reflexive.
Let $aR_{\mathcal{H}}b$. Then $a$ and $b$ belong to the same set $U \in \mathcal{H}$ and hence $bR_{\mathcal{H}}a$.
Thus the relation is symmetric.
Suppose now $aR_{\mathcal{H}}b$ and $bR_{\mathcal{H}}c$. There exists sets $U$ and $V \in \mathcal{H}$ such that $a$ and $b \in U$, and $b$ and $c \in V$. Because $b \in U \cap V$, we have $U \cap V \neq \emptyset$ and hence $U = V$, because $\mathcal{H}$ is disjoint. Thus $a$ and $c$ belong to the same
set $U = V \in \mathcal{H}$, and hence $aR_{\mathcal{H}}c$. The relation $R_{\mathcal{H}}$ is thus transitive and it is an equivalence relation.

Let $R_{\mathcal{H}}(a)$ be an arbitrary $R_{\mathcal{H}}$-equivalence class. Because the union of the sets in $\mathcal{H}$ is $X$, there exits a $U \in \mathcal{H}$ such that $a \in U$. If $x \in U$, then $a$ and $x$ belong to the same set $U \in \mathcal{H}$, hence $aR_{\mathcal{H}}x$ or $x \in R_{\mathcal{H}}(a)$. Conversely, suppose $x \in R_{\mathcal{H}}(a)$ or $aR_{\mathcal{H}}x$. Then $a$ and $x$ belong to the same set $V \in \mathcal{H}$. Because $a \in U \cap V$, we have $U \cap V \neq \emptyset$, and furthermore $U = V$, because $\mathcal{H}$ disjoint. Hence $x \in V = U$. The results show that $R_{\mathcal{H}}(a) = U$. Conversely, if $U \in \mathcal{H}$, then $U \neq \emptyset$. If $a \in U$, then, because of the previous, $R_{\mathcal{H}}(a) \in \mathcal{H}$. Because $R_{\mathcal{H}}(a) \cap U \neq \emptyset$, we have $U = R_{\mathcal{H}}(a)$. Thus the union of the $R_{\mathcal{H}}$-equivalence classes is the same as $\mathcal{H}$. □
Let $R$ be a relation in a set $V$. Define the powers of a relation as follows:

\[
R^0 = \{(a, a) \mid a \in V\}, \\
R^1 = R, \\
R^2 = \{(a, c) \mid \exists b \in V : aRb \text{ ja } bRc \}, \\
R^n = R(R^{n-1}), \quad n > 2.
\]

Transitive closure is now defined with the help of the powers of a relation. The reflexive transitive closure of a relation $R$, $R^*$, is the set

\[
R^* = \bigcup_{i=0}^{\infty} R^i,
\]
and *transitive closure* $R^+$ is the set

$$R^+ = \bigcup_{i=1}^{\infty} R^i.$$  

Writing out the formulas recursively we see that $aR^*b$, if there exists elements $a = c_1, c_2, \cdots, c_n = b$ in $V$ such that $c_i Rc_{i+1}$, $i = 1, \cdots, n - 1$.

Relation $R$ in a set $V$ can be represented as a directed graph: The node set of the graph is $V$ and if $aRb$, then $(a, b)$ is an arc in the graph. The transitive closure has an illustrative interpretation in the graph. That is, $R^+$ means all the pairs $(a, b) \in V \times V$ such that there is a path from $a$ to $b$ in the graph $R$. Similarly, $R^* R^+$ with arcs added from every node to itself.

There are many algorithms to compute the transitive closure of a given relation. However, in many situations in verification, it is not necessary to compute a transitive closure beforehand, but *on the fly*. This means that
when one needs to traverse the arcs of the transitive closure starting from one node, a depth-first search is started from this node and it finds all the paths from that node. It may seem that this method is very time consuming, but it behaves very well in practice.
Trace Equivalence I

The simplest process equivalence is based on the comparisons of event sequences. Let $A$ be the set of actions. We assume in what follows that the actions of all the processes belong to this set.

**Definition**

Let $u \in (A\setminus\{\tau\})^*$ be an action sequence. Sequence $u$ is a trace of $P$, if $P \xrightarrow{u} P'$ for some process $P'$. We denote the set of all the traces of $P$ by $\text{tr}(P)$.
**Definition**

Processes $P$ and $Q$ are trace equivalent, $P \approx_{tr} Q$, if $\text{tr}(P) = \text{tr}(Q)$.

Clearly $\approx_{tr}$ is an equivalence relation. It also is compositional with respect to the parallel operator. Thus if $P \approx_{tr} P'$ ja $Q \approx_{tr} Q'$, then $P|[a_1, \cdots, a_n]|Q \approx_{tr} P'|[a_1, \cdots, a_n]|Q'$ (exercise).

If $P \approx_{tr} Q$, then there may be deadlocks in $P$ even if $Q$ is deadlock-free. For example, the processes $P$ and $Q$ below are trace equivalent.
It is generally thought that deadlocks are the most serious errors in distributed systems. That is why the trace equivalence is not always appropriate when comparing a protocol and its service. However, the trace equivalence reveals other types of mistakes quite effectively. In addition, deadlocks are easy to detect already when generating global state graphs. Thus the trace equivalence may be used sometimes. We will see its usefulness when analysing mutual exclusion algorithms. Furthermore, the trace equivalence is a starting point for the whole family of so called decorated trace equivalences which includes failure and test equivalences.
The bisimulation equivalence was invented by Robin Milner and further developed by David Park at the end of the 70’s and early 80’s. If $P$ and $Q$ are processes, then the idea of the equivalence is to simulate the behaviour of visible actions in $P$ by $Q$ and vice versa. If the simulation succeeds all the time, the processes are equivalent. In order to define this precisely, we need auxiliary concepts.
Weak Bisimulation Equivalence II

Let $a$ be an action, $a \neq \tau$. Remember the notation

$$P \xrightarrow{a} P',$$

which means that there is a transition chain

$$P = P_1 \xrightarrow{\tau} P_2 \xrightarrow{\tau} \cdots \xrightarrow{\tau} P_k \xrightarrow{a} P_{k+1} \xrightarrow{\tau} P_{k+2} \xrightarrow{\tau} P_{k+3} \xrightarrow{\tau} \cdots \xrightarrow{\tau} P_{k+m} = P',$$

$k \geq 1, m \geq 0$. In other words, $P \xrightarrow{a} P'$, if there is a path from the initial state of $P$ to the initial state of $P'$ and one of the arcs belonging to the path contains $a$, and others $\tau$. We can also write

$$P \xrightarrow{\varepsilon} P'$$

if $P = P'$ or there is a chain of $\tau$ transitions

$$P = P_1 \xrightarrow{\tau} P_2 \xrightarrow{\tau} \cdots \xrightarrow{\tau} P_k = P',$$

$k > 1$. 
Example. Consider the following transition system.

There are the following paths from $P_1$: $P_1 \xrightarrow{\varepsilon} P_6$, $P_1 \xrightarrow{a} P_2$, $P_1 \xrightarrow{a} P_5$, $P_1 \xrightarrow{a} P_7$, $P_1 \xrightarrow{b} P_7$, $P_1 \xrightarrow{\varepsilon} P_3$, $P_1 \xrightarrow{\varepsilon} P_1$. □
Definition. Let $P$ and $Q$ be processes and $A$ the action set of $P$ and $Q$. Processes $P$ and $Q$ are weakly bisimilar, $P \simeq_{wbis} Q$, if there is a set $\mathcal{R}$, weak bisimulation, consisting of process pairs such that for every action $a \in (A\setminus\{\tau\}) \cup \{\varepsilon\}$:

1. $(P, Q) \in \mathcal{R}$;
2. if $(P_1, Q_1) \in \mathcal{R}$ ja $P_1 \xrightarrow{a} P_2$, there exists $Q_2$ such that $Q_1 \xrightarrow{a} Q_2$ ja $(P_2, Q_2) \in \mathcal{R}$;
3. if $(P_1, Q_1) \in \mathcal{R}$ ja $Q_1 \xrightarrow{a} Q_2$, there exists $P_2$, such that $P_1 \xrightarrow{a} P_2$ ja $(P_2, Q_2) \in \mathcal{R}$.
Example. The following processes are weakly bisimilar, $P \approx_{wbis} Q$:

Equivalence follows, because 
$\mathcal{R} = \{(P_1, Q_1), (P_2, Q_2), (P_3, Q_3), (P_4, Q_1)\}$ is a weak bisimulation and 
$(P, Q) \in \mathcal{R}$ ($P = P_1$, $Q = Q_1$). □
Example. The following processes are not weakly bisimilar:

If we try to construct a weak bisimulation $R$, then the pair $(P1, Q1)$ must belong to that relation. Let us use next the condition 3 in the definition: $Q$ makes a transition from $Q1$ to $Q2$ using an internal action. The only way to simulate this transition in $P$ is such that we must stay in $P1$. Hence the pair $(P1, Q2)$ must belong to the bisimulation relation $R$. After
this, we apply the condition 2 to the pair \((P_1, Q_2)\): \(P\) moves from \(P_1\) with \(a\) to \(P_2\). Now \(Q\) cannot simulate this transition, because there are no \(a\)-transitions from \(Q_2\). Thus there cannot exists a weak bisimulation between \(P\) and \(Q\) and hence the processes are not bisimilar. \(\square\)
Example. The following processes are weakly bisimilar:
The weak bisimulation is as follows:

\[(P_1, Q_1), (P_2, Q_2), (P_4, Q_3), (P_6, Q_4),
(P_3, Q_5), (P_4, Q_6), (P_5, Q_7), (P_6, Q_8), (P_7, Q_9)
(P_5, Q_{10}), (P_6, Q_{11}), (P_7, Q_{12}), (P_7, Q_{13})\]
\( \approx_{wbis} \) is an equivalence relation I

**Theorem.** *The relation \( \approx_{wbis} \) is an equivalence relation between transition systems.*

**Proof.** We must show that the relation \( \approx_{wbis} \) is reflexive, symmetric, and transitive.

- If \( \approx_{wbis} \) is reflexive, then we should have \( P \approx_{wbis} P \) for all processes \( P \).
- Symmetry means that from \( P \approx_{wbis} Q \) it follows that \( Q \approx_{wbis} P \). Both properties follow directly from the definition.
- We still must show that the relation is transitive, i.e. from the conditions \( P \approx_{wbis} Q \) ja \( Q \approx_{wbis} R \) it follows that \( P \approx_{wbis} R \). Let \( \mathcal{R} \) a bisimulation between \( P \) and \( Q \), \( \mathcal{S} \) a bisimulation between \( Q \) and \( R \).
- Construct the set \( \mathcal{T} \) of process pairs as follows:

\[
\mathcal{T} = \{(P_1, R_1) | \exists Q_1 : (P_1, Q_1) \in \mathcal{R}, (Q_1, R_1) \in \mathcal{S}\}.
\]
Let us show that $T$ is a weak bisimulation between $P$ and $R$. Because of the definition of $T$ we have $(P, R) \in T$. Let $(P_1, R_1) \in T$ an arbitrary element and $P_1 \xrightarrow{a} P_2$.

We know that there exists $Q_1$ such that $(P_1, Q_1) \in R$ ja $(Q_1, R_1) \in S$.

Because $R$ and $S$ are weak bisimulations, there exist processes $Q_2$ and $R_2$ such that $Q_1 \xrightarrow{a} Q_2$ and $(P_2, Q_2) \in R$, and further $R_1 \xrightarrow{a} R_2$ ja $(Q_2, R_2) \in S$. But by the definition of $T$ we have $(P_2, R_2) \in T$, and thus the condition 2 has been shown.

The condition 3 is shown in the same way. □
Denote by $\mathcal{P}$ the set of all finite processes or labelled transition systems. Relation $\approx_{wbis}$ thus defines an equivalence relation in $\mathcal{P}$.

In mathematics, an equivalence relation is defined as a subset of the cartesian product of the basic set with itself. Thus in our case it should be that $\approx_{wbis} \subseteq \mathcal{P} \times \mathcal{P}$. We could indeed define that $\approx_{wbis}$ is the maximal bisimulation

$$\approx_{wbis} = \bigcup \{ \mathcal{R} \mid \mathcal{R} \text{ on bisimulation} \}.$$ 

If $P$ is a process, then the equivalence class $[P]_{\approx_{wbis}}$ of $P$ is the set of those processes equivalent with $P$. The classes $[P]_{\approx_{wbis}}$ form a partition of $\mathcal{P}$. 
If $P$ is a labelled transition system $(S, A, \rightarrow, s_0)$, then every state $s \in S$ can be considered as a transition system which is the same as $P$, but the initial state is now $s$.

It is now possible to restrict the equivalence $\approx_{wbis}$ to the set $S \times S$. Then $\approx_{wbis}$ is an equivalence relation in $S$ and it partitions $S$ into equivalence classes.

We can now form a new transition system whose states are the equivalence classes. There is an arc with $a$ from one class to another, if there is a state $s_1$ in the first class and a state $s_2$ in the second class and an arc $s_1 \xrightarrow{a} s_2$ in the original transition system.

The transition system constructed in this way is the minimal transition system among those systems that are weak bisimulation equivalent with the original one.
For example, the following processes are equivalent and the latter is minimal.
Minimal Process III

\[ P: \]

\[ P_{1} \rightarrow a \rightarrow P_{2} \rightarrow \tau \rightarrow P_{3} \rightarrow b \rightarrow P_{4} \rightarrow \tau \rightarrow P_{5} \]

\[ P_{6} \rightarrow \tau \rightarrow P_{2} \rightarrow \tau \rightarrow P_{7} \rightarrow \tau \rightarrow P_{4} \]

\[ P_{7} \rightarrow \tau \rightarrow P_{5} \]

\[ P_{min}: \]

\[ P_{1} \rightarrow a \rightarrow P_{2} \rightarrow \tau \rightarrow P_{3} \]

\[ \tau \rightarrow P_{2} \rightarrow b \rightarrow P_{3} \]
The equivalence classes of $P$ are $E_1 = \{P1, P5\}$, $E_2 = \{P2, P4, P6, P7\}$ ja $E_3 = \{P3\}$.

If we draw all the transitions mechanically from one class to another, the result may contain too many useless arcs. However, the minimization of arcs is more complicated than that of states. More details can be found in the dissertation of Jaana Eloranta and in the article Eloranta, Tienari, Valmari: Essential Transitions To Bisimulation Equivalences, Theoretical Computer Science 179 (1997) 397-419.
The weak bisimulation equivalence behaves well with respect of the parallel operator.

**Theorem.** If $P \approx_{wbis} Q$, then $P \parallel B \parallel R \approx_{wbis} Q \parallel B \parallel R$ for all action sets $B$ and processes $R$.

**Proof.** Let

$$R = \{(P_1 \parallel B \parallel P_3, P_2 \parallel B \parallel P_3) \mid P_1 \approx_{wbis} P_2\}.$$ 

Let us show that $R$ is a weak bisimulation between $P \parallel B \parallel R$ and $Q \parallel B \parallel R$. Because $P \approx_{wbis} Q$, we have $(P \parallel B \parallel R, Q \parallel B \parallel R) \in R$. Let $P \parallel B \parallel R \xrightarrow{a} P' \parallel B \parallel R'$. We have to check two cases.
Weak Bisimilarity and the Parallel Operator II

i) \( a \in B \). Now \( a \neq i \) and \( P \xrightarrow{a} P' \), \( R \xrightarrow{a} R' \). Because \( P \approx_{wbis} Q \), there exists a weak bisimulation \( \mathcal{E} \) between \( P \) and \( Q \). We know, because of the definition of the weak bisimulation, that there exists \( Q' \) such that \( Q \xrightarrow{a} Q' \) and \( (P', Q') \in \mathcal{E} \). Hence also \( P' \approx_{wbis} Q' \). We can directly see that \( Q|B|R \xrightarrow{a} Q'|B|R' \). The definition of \( \mathcal{R} \) implies now that \( (P'|B|R', Q'|B|R') \in \mathcal{R} \).

ii) \( a \notin B \). Now either \( P \xrightarrow{a} P' \) and \( R \xrightarrow{a} R' \) or \( P \xrightarrow{a} P' \) and \( R \xrightarrow{a} R' \). Notice that \( a \) can be \( \varepsilon \). If \( P \xrightarrow{a} P' \), then also \( Q \xrightarrow{a} Q' \) and \( P' \approx_{wbis} Q' \), as we saw in the previous case. Thus \( Q|B|R \xrightarrow{a} Q'|B|R' \) and \( (P'|B|R', Q'|B|R') \in \mathcal{R} \). If, on the hand, \( R \xrightarrow{a} R' \), \( P \xrightarrow{a} P' \), then there also exists \( Q' \) such that \( Q \xrightarrow{a} Q' \) and \( P' \approx_{wbis} Q' \). Furthermore \( Q|B|R \xrightarrow{a} Q'|B|R' \). Now it is true that \( (P'|B|R', Q'|B|R') \in \mathcal{R} \).
Thus $\mathcal{R}$ satisfies the conditions of the weak bisimulation as for the transitions of in $P|B|R$. The transitions in $Q|B|R$ are handled in the same way and we can conclude that $\mathcal{R}$ is a weak bisimulation. $\square$
Consider the version of the AB-protocol where there are the messages get and give. This version defines a service which is easily described:

\[ \text{give} \quad P1 \quad P2 \]

\[ \text{get} \]
Let us apply the weak bisimulation equivalence to check the correctness of the protocol. Thus we have to show that the global state graph of the protocol is equivalent with the service. Evidently this is not true, if we do not change the global state graph in some way. One usual way is to hide some actions. For this purpose we define the operation hide:

\[
\text{hide } a_1, a_2, \ldots, a_n \text{ in } P
\]

transforms \( P \) in such a way that all the actions \( a_i, i = 1, \ldots, n, \) in \( P \) are replaced with \( \tau \).
Now we can phrase the verification problem of the AB-protocol in the following form. We have to show that

\[
\begin{align*}
\text{hide } & d0, dd0, d1, dd1, a0, aa0, a1, aa1, st, rt, t \text{ in AB-protocol} \\
\approx_{wbis} & \text{ AB-service.}
\end{align*}
\]

The global state graph of the AB-protocol is a little too large for a manual construct so that it is necessary to use software to generate it. In order to do this, we must first write the protocol and its service in some specification language. After this the software generates the global state graphs of the protocol and service and finally compares these two to check if they are equivalent or not. We will do all this after we have studied Lotos.
In the old protocol articles there was one erroneous protocol which has been in practical use. (W. C. Lynch: Reliable full-duplex transmission over half-duplex lines, Comm. ACM, Vol. 11, No. 6, pp 362-372, June 1968).

Its mistakes appeared so seldom that the errors were not observed in testing, but sometimes they were encountered in the production use. It is a protocol that is illustrative to analyse.

Our protocol, shortly FE-protocol, is a symmetric link layer protocol. Two participants $S$ and $R$ change messages alternatively using a half-duplex channel. Acknowledgements $a$ (positive acknowledgement) are added into every message and this tells that the previous message has arrived correctly. The acknowledgement is $n$ (negative acknowledgement), if the previous message was distorted in the channel.
Example: FE-protocol II

- The sending logic was as follows: If the previous message contained \( n \) or it was distorted (maybe \( n \) was destroyed), send the earlier message again. Otherwise send a new message. In both cases ACK or NAK is added into the message according to the situation.

- The receiving logic is not given explicitly in the article. The article shows only that it is impossible to design a receiving logic so that the protocol works correctly. We must use some logic and one possible logic would be as follows: When a message arrives correctly and contains ACK, it is delivered to the client. If a message contains NAK, the message is not delivered further (it is assumed that the message is a repetition of an older message).

- First we build the modell of the protocol in a simplified form. If the protocol behaves incorrectly in this model, it behaves incorrectly in a more general model, too. If, on the other hand, the simplified version works correctly, it is still necessary to check the behaviour of the more general version.
In the simplified version the communication is synchronous, we do not use separate channels. Data is sent only from $S$ to $R$. The notation $da$ means data with a positive acknowledgement, and $dn$ is data with a negative acknowledgement. $R$ will send only acknowledgements $a$ or $n$. The notations $de$ and $e$ represent distorted data and acknowledgement, respectively. The labelled transition graphs $S$ and $R$ are given below:
Example: FE-protocol IV
The global state graph is in the following diagram:
We can see the basic cycle of the protocol,
which describes the messages when the communication is error-free. The protocol recovers from a single communication error using an additional path:

\[
S_2R_3 \xrightarrow{de} S_3R_4 \xrightarrow{n} S_2R_3
\]

or

\[
S_3R_2 \xrightarrow{e} S_4R_3 \xrightarrow{dn} S_3R_2
\]

Two sequential communication errors may cause that the protocol does not behave correctly. This we can see, if we traverse the following path in the global state graph:
In this scenario two data messages are sent, but only one of them is really delivered to the receiver. The protocol may thus lose messages. Moreover, it is possible that the protocol delivers the same message to the receiver:
The former analyses were based on a detailed scrutiny of the global state graph.

This erroneous behaviour can also be detected automatically. For this purpose we need the service description which happens to be the same as in the AB-protocol.

With the help of a software we can show that

\[
\text{hide } da, de, a, n, e \text{ in } \text{FE-protocol } \not\approx_{wbis} \text{FE-service.}
\]

It is easy to see this manually, too. Let us try to construct a weak bisimulation:

\[
\mathcal{R} = \{(S1R3, P1), (S2R3, P2), (S3R2, P2), (S1R3, P2) \ldots \}.
\]

Now get can be done in state $S1R3$, but in $P2$ only give is possible. Thus it is not possible to construct a weak bisimulation and the processes are not equivalent.
It is possible to develop the formalisms based on labelled transition systems further. If we will apply these methods in practice, we must pay attention on the following points:

1. How are labelled transition systems represented? Diagrams are not suitable for computers.

2. How is the collaboration of several processes described? We must decide the following questions:
   - how is time modelled;
   - is the communication synchronous or asynchronous;
   - is multisynchronization allowed;

3. How is data taken into account in the messages?

4. Transitions may depend on the content of data. How is this handled?

5. In the modelling of real time systems we need time constraints. How are these expressed in the formalism?
Transitions may have completely different probabilities (for example, the probability of a communication error may be very small). Is it wise to assume that all the transitions happen with the same probability?

In ordinary transition systems synchronization points (messages) are known beforehand and their use is fixed when the specification is written. There are, however, a lot of applications, where the synchronization points are dynamically created. For example, when one needs services, it may be that the address or permit for the service is obtained first from some authority, who answers by giving the port or address of the service. How are these kinds of situations taken into account?
All the previous tasks have been implemented in one form or another. Especially, the items 1-4 have been solved in a generally accepted way. Labelled transition systems are usually described with the help of process algebras, for example Milner’s CCS, Hoare’s CSP, and Bergstra’s and Klop’s ACP. In these, the items 1) and 2) have been solved in a similar way:

- A labelled transition system is described using algebraic expressions.
- Cycles are generated with the help of recursion. Recursion also enables the definitions of an infinite systems. Furthermore, it makes possible to start processes dynamically.
- Processes communicate synchronously. It means that a process cannot finish the sending of a message before the receiver has really received the message. Asynchronous communication is achieved by using separate channel processes as we saw in the AB-protocol.
- Concurrent actions are performed sequentially using interleaving:
Events are atomic.
If actions $a$ and $b$ are executed concurrently, then we think that either $a$ happens before $b$ or $b$ before $a$. Thus we have two possible execution sequences $ab$ and $ba$ which are present in the global state graph of a system.

Because concurrency is modelled using interleaving, there are a lot of different alternatives for execution sequences. Thus global state graphs tend to be large (abc, acb, cab, bac, bca, cba).

In CCS, only two processes can synchronize with each other whereas in CSP and ACP multi-synchronization is possible.

In process algebras, it is possible to state conditions for transitions. On the other hand, it is not possible directly to express real time constraints or probabilities in these traditional process algebras. There are newer formalisms which tackle these questions. Lotos has many features from CCS and CSP. It has

- synchronous communication,
Conclusions and Problems V

- multi-synchronization,
- interleaving semantics.

a) In Lotos, it is possible to combine processes in a more general way than in the previous process algebras. This causes both advantages and disadvantages. For example, in the previous process algebras it is possible to prove useful algebraic properties for parallel operators. Lotos has not these properties (for example the operator is not associative).

b) The representation of data is the most distinct feature in Lotos. In Lotos, data is defined using the algebraic specification of abstract data types. This algebraic specification style was developed since 1970 and it was closely connected to denotational semantics. In algebraic specification, the meaning, or semantics, is created by defining relations the operators satisfy. This is a very
powerful technique. For example, it is possible to defined natural numbers without basing the definition on any other constructions. On the other hand, the data types thus defined are very inefficient (for example, number 3 is $\text{succ}(\text{succ}(\text{succ}(0)))$). In this course, we skip the data part of Lotos for the most part.

c) Lotos is practice-oriented. It is larger than the theoretical languages CCS, CSP, and ACP. Especially the data definition part of Lotos is different. Even if we concentrate solely on basic Lotos, it is not wise to use only that part in practice. In practical verifications, full Lotosa should be used, because in this way the descriptions become clearer and more concise.
d) There are also extensions of Lotos (E-Lotos). They include real time properties. Furthermore, dynamical synchronization is essential and there have been suggestions to include it into Lotos, but the results have not been successful. Instead, CCS has been extended to include dynamical synchronization and this extension is known by name $\pi$-calculus.

e) There are many more process algebras nowadays. They have been designed for specific purposes. *Ambient calculus* describes concurrent systems with mobility. There can be two kinds of mobility: devices move or code moves. Ambient calculus can handle both. *PEPA* is a stochastic process algebra which has been designed for modelling computer and communication systems. It has probabilistic branching and transitions may have timed conditions. *Fusion calculus* is a modification of $\pi$-calculus. *Spi-calculus*, also a modification
of $\pi$-calculus, has been designed for the verification and analysis of security properties.

Finally, we should mention Robin Milner’s latest invention, bigraphs. They are designed to be a platform for ubiquitous computing systems. Bigraphs are powerful objects and it is possible to describe various structures, even biological structures, with the help of them.