Before defining global state graphs formally we show a concrete example based on the AB-protocol. In this protocol, sender $S$ sends messages $d_0$ and $d_1$ to receiver $R$. Receiver $R$ sends acknowledgements $a_0$ and $a_1$ to $S$. Suppose that the communication between $S$ and $R$ is rendezvous-type. Thus before a sender can send, a receiver must be ready to receive. If a receiver is not ready, a sender cannot send. For example, if $S$ sends $d_0$, $S$ moves to a next state with the action $d_0$ and at the same time $R$ moves to the next state with $d_0$.

The internal transition $\tau$ does not cause other processes to change their states. Similarly, the timer action $t$ changes only the state of the sender.

A state in the global state graph is a pair $(S_i, R_j)$, where $S_i$ is a state of the sender and $R_j$ is a state of the receiver. Let us draw now the global state graph. We make no assumptions about the timer. It may timeout too early, but this should cause no problems.
The Global State Graph of the AB-protocol II
Next we define the global state graph formally with the help of the parallel operator. Let $P$ ja $Q$ be transition systems.

Suppose that the states in $P$ are $P_1, P_2, \cdots, P_m$ and the states in $Q$ are $Q_1, Q_2, \cdots, Q_n$. The initial state of the whole system is $(P_1, Q_1)$, where $P_1$ is the initial state in $P$ and $Q_1$ is the initial state in $Q$.

The states in the global state graph are of the form $(P_i, Q_j)$, $i = 1, \cdots, m$, $j = 1, \cdots, n$.

We write $P_i \xrightarrow{a} P_i'$, if there is a transition from $P_i$ to $P_i'$ with an action $a$. Similarly in the case of $Q$.

The global state graph $P|[[a_1, \cdots, a_k]]Q$ of the processes $P$ and $Q$ is defined now formally by giving the rules which show how to move from one state to another.
Parallel Operator II

- The actions $a_1, \cdots, a_k$ are *synchronizing actions*. If one process is to perform $a_i$, then the other synchronizes and performs it, too. If the other cannot perform $a_i$, then neither the first nor the second can perform it.

- Other actions can and must be performed without synchronization.

- We define now the transitions between the states. This is done with the help of a so called parallel operator $|[a_1, \cdots, a_k]|$.

- It is demanded that $\tau \neq a_i$ for all $i = 1, \cdots, k$.

- If we apply the parallel operator to the state pair $(P_i, Q_j)$, the synchronizing actions are marked by writing $P_i|[a_1, a_2, \cdots, a_k]|Q_j$.

- The following rules define what transitions are possible from a given global state, i.e. from a state pair $(P_i, Q_j)$. 
If $a \in \{a_1, \cdots, a_k\}$, $P_i \xrightarrow{a} P_i'$ and $Q_j \xrightarrow{a} Q_j'$, then

$$P_i|[a_1, a_2, \cdots, a_k]|Q_j \xrightarrow{a} P_i'|[a_1, a_2, \cdots, a_k]|Q_j'.$$

If $a \not\in \{a_1, \cdots, a_k\}$ and $P_i \xrightarrow{a} P_i'$, then

$$P_i|[a_1, a_2, \cdots, a_k]|Q_j \xrightarrow{a} P_i'|[a_1, a_2, \cdots, a_k]|Q_j.$$
The application determines what is a suitable synchronizing set. If \( k = 0 \), i.e. no action synchronizes, we speak about complete interleaving and use the notation \(|||\). If the synchronizing set consists of all the visible actions, we use the notation \(||\).

The final global state graph consists only of those states that can be reached from the initial state. If we use the parallel operator to construct the global state graph of the AB-protocol, we start from the formula

\[
S|[d0, d1, a0, a1]|R
\]

and deduce all the other states and transitions from this formula. The result as a graph is the same as before.
Example 1 |
Example 2

\[ P: \quad P1 \xrightarrow{a} P2 \]

\[ Q: \quad Q1 \xrightarrow{b} Q2 \]

\[ P \parallel [a, b] \parallel Q: \]

\[ P1 \quad Q1 \]
Multisynchronization I

It is possible to synchronize several processes at the same time. For example, in the formula

$$P \mid [a] \mid (Q \mid [a] \mid R)$$

the action $a$ can happen in $Q$ and $R$ only if both processes perform that action at the same time. On the other hand,

$$Q \mid [a] \mid R$$

is a process, too, and thus $a$ can happen in it and in $P$ only if all the three processes perform it at the same time.
Synchronization is *symmetric*. We do not distinguished which process starts it. Thus the sender and receiver are equal when a message is sent: both must be ready and perform an action.

Synchronization is *nameless*. If a process offers synchronization, any process with an suitable action may take part in the synchronization. It is not possible for a process to demand that a synchronization is directed to a particular process. This property has advantages and drawbacks in applications.
In general, the parallel operator is not associative. Thus it is not that

\[ P | A_1 | (Q | A_2 | R) = (P | A_1 | Q) | A_2 | R. \]

In the following cases the parallel operator is, however, associative:

**CSP’s case.** Let \( P, Q \) and \( R \) be processes and \( A_P, A_Q, A_R \) the action sets of the processes. Then

\[ P | A_P \cap (A_Q \cup A_R) | (Q | A_Q \cap A_R | R) \equiv (P | A_P \cap A_Q | Q) | (A_P \cup A_Q) \cap A_R | R, \]

where ‘\( \equiv \)’ means that the corresponding transition systems are the same, only the names of the states are different (notice that when using the parallel operator state in a global state graph contains that operator).
If $B$ is an arbitrary action set, then

$$P \mid B \mid (Q \mid B \mid R) \equiv (P \mid B \mid Q) \mid B \mid R.$$  

With synchronizing sets $B_1$ and $B_2$ we have

$$P \mid B_1 \mid (Q \mid B_2 \mid R) \equiv (P \mid B_1 \mid Q) \mid B_2 \mid R,$$

if $A_P \cap B_2 = \emptyset$ and $A_R \cap B_1 = \emptyset$.

*Proof of the item 1.* It is enough to prove that every transition in

$$P \mid A_P \cap (A_Q \cup A_R) \mid (Q \mid A_Q \cap A_R \mid R)$$

corresponds exactly the same transition in

$$(P \mid A_P \cap A_Q \mid Q) \mid (A_P \cup A_Q) \cap A_R \mid R$$
and vice versa. It is possible to follow two strategies in the proof. In the first strategy we examine a single transition with \(a\) and deduce what action sets \(a\) belongs to. In the other strategy we examine what components in the system change their states and deduce what kind of transition takes place in the other system. Let us follow the second strategy.

In what follows we denote by a apostrophe the process that changes in the transition. We have to check several alternatives.

a) \[ P | A_P \cap (A_Q \cup A_R) | (Q | A_Q \cap A_R | R) \xrightarrow{a} P' | A_P \cap (A_Q \cup A_R) | (Q | A_Q \cap A_R | R). \]
b) \[ P | A_P \cap (A_Q \cup A_R) | (Q | A_Q \cap A_R | R) \xrightarrow{a} P | A_P \cap (A_Q \cup A_R) | (Q' | A_Q \cap A_R | R). \]
c) \[ P | A_P \cap (A_Q \cup A_R) | (Q | A_Q \cap A_R | R) \xrightarrow{a} P | A_P \cap (A_Q \cup A_R) | (Q | A_Q \cap A_R | R'). \]
d) \[ P | A_P \cap (A_Q \cup A_R) | (Q | A_Q \cap A_R | R) \xrightarrow{a} P' | A_P \cap (A_Q \cup A_R) | (Q' | A_Q \cap A_R | R). \]
e) \[ P | A_P \cap (A_Q \cup A_R) | (Q | A_Q \cap A_R | R)^a \rightarrow P' | A_P \cap (A_Q \cup A_R) | (Q | A_Q \cap A_R | R'). \]

f) \[ P | A_P \cap (A_Q \cup A_R) | (Q | A_Q \cap A_R | R)^a \rightarrow P | A_P \cap (A_Q \cup A_R) | (Q' | A_Q \cap A_R | R'). \]

g) \[ P | A_P \cap (A_Q \cup A_R) | (Q | A_Q \cap A_R | R)^a \rightarrow P' | A_P \cap (A_Q \cup A_R) | (Q' | A_Q \cap A_R | R'). \]

Case a) In this case the transition takes place only inside \( P \). Thus \( a \in A_P \) ja \( a \notin A_Q \cup A_R \). Hence \( a \notin A_P \cap A_Q \) and \( a \notin (A_P \cup A_Q) \cap A_R \), and we can deduce that the transition happens also in 
\((P | A_P \cap A_Q | Q) | (A_P \cup A_Q) \cap A_R | R\) only in \( P \). And the transition is the same.

Cases b) and c) are proved in the same way as a).

Case d) Now the transition happens both in \( P \) and in \( Q \). Thus \( a \in A_P \) and \( a \in A_Q \cup A_R \), but \( a \notin A_Q \cap A_R \), hence \( a \in A_Q \) and \( a \notin A_R \). We can deduce
that \( a \in A_P \cap A_Q \), but \( a \not\in (A_P \cup A_Q) \cap A_R \). Because of this, the transition with \( a \) in the global state graph \((P \mid A_P \cap A_Q \mid Q) \mid ((A_P \cup A_Q) \cap A_R) \mid R\) happens only in \( P \) and \( Q \), and they change in the same way as in the graph \( P \mid A_P \cap (A_Q \cup A_R) \mid (Q \mid A_Q \cap A_R \mid R) \).

**Cases e) ja f)** are proved in the same way as d).

**Case g)**. Now all the processes change their states and thus \( a \in A_P \cap (A_Q \cup A_R) \) ja \( a \in A_Q \cap A_R \). Hence \( a \in A_P \), \( a \in A_Q \) and \( a \in A_R \). Furthermore, \( a \in A_P \cap A_Q \) and \( a \in (A_P \cup A_Q) \cap A_R \). Thus the transition takes place in the graph \((P \mid A_P \cap A_Q \mid Q) \mid ((A_P \cup A_Q) \cap A_R) \mid R\) in all the processes and exactly in the same way as in \( P \mid A_P \cap (A_Q \cup A_R) \mid (Q \mid A_Q \cap A_R \mid R) \). □

Item 2) is proved in the same way as 1). Item 3) is a modification of item 1).
The global state graph is used in two ways. In some applications, it is enough to traverse the graph without constructing it completely. In some other applications, it is necessary to construct the whole graph explicitly. In this latter alternative there is a problem that the graph may be too large. As matter of fact, many practical protocols and systems lead to such a large graph that it is not possible to generate it completely. For example, it has not been easy to analyse formally the collaboration of several layers in network environments.

The global state graph is usually sparse. Thus there are only few out-going arcs from the states. Hence matrix representations are not promising, but adjacency list representations perform better. Usually the generation and analysis is done depth-first. Thus we start from the initial global state and generate all the states one step away from the initial state. These new states are pushed into a stack. Then continue as follows as long as the stack is non-empty: Take a state from the stack, generate all the
neighbours of the chosen state, push the neighbours into the stack and draw the arcs from the chosen state to the neighbours. It is necessary to check always during the generation of new states, if the new states are really new or already generated. Usually this check is done with the help of hashing.
Because a global state graph is often large, we need a large hash table, too. It would be tempting to use the virtual memory, but this brings problems. The reason is that during the generation of states new and old states come in unpredictable order. This leads to the fact that it is necessary to fetch pages continually from the disk. This makes the processing too slow. That is why one tries to use only the central memory.

The solution of G. Holzmann was to use bit hashing. In this method, a global state is interpreted as a bit string and furthermore as an integer. Reserve now a boolean array, whose size is such that the index space includes the largest integer corresponding a global state. This kind of an array can be used efficiently for hashing and one state demands only one bit.

The whole transition system can be represented in an alternative way. Boolean functions can be represented in a compact way and transition system can be represented with the help of boolean functions. This
representation is called BDD or binary decision diagram. Neither is this solution universal: in some situations BDD’s help to compress the global state graph, but not always. Moreover, many algorithms work easier with adjacency lists than with BDD’s.