Chapter 2

Protocol design and analysis: the CFSM model

In this chapter we present a simple model for the specification and verification of communication protocols: the communicating finite state machine (CFSM) model. The CFSM model lends itself to reachability analysis, a state exploration technique which has been advocated for verifying general properties of protocols such as absence of deadlocks, non-executable transitions, unspecified receptions and unbounded channel growth [ZW+80, BZ83]. We address two principal limitations of reachability analysis, viz. undecidability and state explosion.

2.1 Modeling protocols as networks of CFSMs

A communication protocol consisting of interacting processes can be modeled as a network of communicating finite state machines (CFSMs). Each CFSM is an abstraction of one of the processes in the protocol and communicates with the other CFSMs by sending and receiving messages over error-free simplex channels, represented by FIFO queues.

Notation 2.1
Given a set $A$, $|A|$ denotes its cardinality and $A^*$ denotes the set of strings of elements in $A$, including the empty string $\square$. Juxtaposition is used to denote concatenation of strings. For a string $Y \sqcup A^*$, $|Y|$ denotes its length and $\text{front}(Y)$ denotes the first element of $Y$; $\text{front}(Y)$ is undefined if $|Y| = 0$. Also, $\underbrace{Y_1 \sqcup \cdots \sqcup Y_n}_{|A|}$ denotes an $|A|$-tuple $(a_{i_1}, a_{i_2}, \ldots, a_{i_n})$.

Definition 2.2
Let $I = \{1, 2, \ldots, n\}$ be a finite index set with $n \geq 2$. A protocol $\square$ is a pair $(P, L)$, where

- $P = \{P_i \mid i \in I\}$ is a set of $n$ processes,
- $L \sqcup I \sqcup I$ is an irreflexive incidence relation identifying a nonempty set of error-free simplex channels $\{C_{ij} \mid (i, j) \in L\}$.
Each process $P_i \sqsubseteq P$ is a quadruple $(S_i, s^0_i, M_i, \partial_i)$, where

- $S_i$ is a finite, nonempty set of process states,
- $s^0_i \sqsubseteq S_i$ is the initial state of $P_i$,
- $M_i = \bigcup_{j \neq i} (M_{ij} \sqsubseteq M_{ji})$, with $M_{ij}$ a finite set of messages that $P_i$ can send to process $P_j$,
- $\partial_i = \bigcup_{j \neq i} \partial_{ij}$ is a finite set of transitions, with $\partial_{ij} \sqsubseteq S_i \sqsubseteq \tilde{M}_{ij} \sqsubseteq S_i$ and $\tilde{M}_{ij} = \{ -x \mid x \sqsubseteq M_{ij} \}$ (plus $+y \mid y \sqsubseteq \tilde{M}_{ij}$),

such that $\partial_{jkl}$, $k, l \sqsubseteq I$: (i) $S_j \sqsubseteq S_j = \emptyset$ if $i \neq j$, (ii) $M_{ij} \sqsubseteq M_{kl} = \emptyset$ if $(i, j) \neq (k, l)$ and (iii) $M_{ij} = \emptyset$ if $(i, j) \sqsubseteq L$.

Each error-free simplex channel $C_{ij}$ ($(i, j) \sqsubseteq L$) is a perfect FIFO queue linking process $P_i$ to process $P_j$. The content $c_{ij}$ of $C_{ij}$ is a string of messages from $M_{ij}$, i.e. $c_{ij} \sqsubseteq M_{ij}^*$. □

A protocol can be viewed as a finite directed graph in which the nodes correspond to processes and the edges correspond to simplex channels. This graph implicitly defines the communication topology of the protocol. Similarly, a process can be viewed as a finite directed graph in which the nodes correspond to process states and the edges correspond to transitions. Each transition represents the occurrence of an event, being either the transmission or the reception of a message by a process. Notice that processes need not be deterministic (i.e. any two transitions of the same process can be such that they differ only in their third component).

**Definition 2.3**

The topology graph of a protocol $\sqsubseteq = (\{ P_i \mid i \sqsubseteq I \}, L)$, denoted by $TG_{\sqsubseteq}$, is the directed graph with vertex set $I$ such that there is an edge from $i \sqsubseteq I$ to $j \sqsubseteq I$ iff $(i, j) \sqsubseteq L$. The process graph of a process $P_i = (S_i, s^0_i, M_i, \partial_i)$ is the labeled directed graph with vertex set $S_i$ such that there is an edge labeled $\partial$ from $S_i$ to $S_i$ iff $(s, \partial, s) \square_i$.

**Definition 2.4**

Let $\sqsubseteq = (\{ P_i \mid i \sqsubseteq I \}, L)$ be a protocol and $t = (s, \partial, s', \partial')$ a transition (at $s$), for some $i \sqsubseteq I$. $t$ is a send transition iff $\partial = -x$, with $x \sqsubseteq \bigcup_{j \neq i} M_{ij}$. $t$ is a receive transition iff $\partial = +y$, with $y \sqsubseteq \bigcup_{j \neq i} M_{ji}$. □

The states of a protocol as a whole, or global states, are defined as the composite of individual (or local) states and channel contents. A global state assigns to each process a process state of this process, and to each simplex channel a sequence of messages in transit over this channel. A special global state is designated as the initial global state.

**Definition 2.5**

Let $\sqsubseteq = (\{ P_i \mid i \sqsubseteq I \}, L)$ be a protocol. A global state $G$ of $\sqsubseteq$ is a pair $(S, C)$, where
\[ S = \bigcup_i G_i \bigcup_{j} I_j \] with \( s_i^G \bigcup S_i \) the process (or local) state of process \( P_i \) in \( G \),

\[ C = \bigcup_i G_i \bigcup_{(i,j)} I_{ij} \] with \( c^{G}_{ij} \bigcup M_{ij} \) the content of simplex channel \( C_{ij} \) in \( G \).

The initial global state, denoted by \( G^0 \), is the global state \( \bigcup_i G_i \bigcup_{(i,j)} I_{ij-L} \) with \( s_i^{G^0} = s_i^0 \) and \( c^{G^0}_{ij} = \emptyset \) for all \( i \bigcup I \) and \( (i,j) \bigcup L \).

Processes can execute transitions at global states. A receive transition defined at \( G \) can be executed when the message to be received resides at the front of the respective queue. Although a send transition could similarly be considered executable if the respective queue is not full, in the traditional CFSM model queues are not a priori bounded\(^\dagger\). A send transition defined at a global state is then always executable.

**Definition 2.6**

Let \( G \) be a global state of a protocol \( \emptyset = (\{P_i \mid i \bigcup I\},L) \) and \( t = (s, \emptyset, s^G \bigcup_{i \bigcup j} I_{ij}) \) a transition, for some \( i, j \bigcup I \). When \( s = s_i^G \), \( t \) is said to be defined at \( G \). \( t \) is executable at \( G \) iff \( t \) is defined at \( G \) and

- \( t \) is a send transition, or
- \( t \) is a receive transition with \( \emptyset = +y \) and \( \text{front}(c^{G}_{ji}) = y \).

The set of send and receive transitions from \( I_{ij} \) that are executable at \( G \) are denoted by \( X_{ij}^S(G) \) and \( X_{ij}(G) \), respectively, and \( X_{ij}(G) = X_{ij}^S(G) \bigcup X_{ij}^R(G) \).

**Notation 2.7**

\[
\begin{align*}
X_i^S(G) &= \bigcup_{j \bigcup i} X_{ij}^S(G) & X_i^R(G) &= \bigcup_{j \bigcup i} X_{ij}^R(G) & X_i(G) &= X_i^S(G) \bigcup X_i^R(G) \\
X^S(G) &= \bigcup_{i \bigcup j} X_{ij}^S(G) & X^R(G) &= \bigcup_{i \bigcup j} X_{ij}^R(G) & X(G) &= X^S(G) \bigcup X^R(G)
\end{align*}
\]

\[ \square \]

### 2.2 Reachability analysis

A protocol defined in the CFSM model is a closed system: starting at the initial global state it can evolve and change its global state by executing transitions. A natural way to analyze the possible behavior of a protocol is then to consider its set of reachable global states and the transitions that occur between them. More specifically, the complete interaction domain of the protocol can be

---

\(^\dagger\) In [BZ83] the use of unbounded channels is justified conceptually as follows: “The queues modeling the simplex channels have unbounded capacity to represent protocols allowing an arbitrary number of messages in transit. In a physical implementation all channels must be bounded, but the bound may be too large to be of practical use. Moreover, since protocols are supposed to operate over different channels with different capacities, a channel of unbounded capacity is the proper abstraction.”
examined by exploring all possible ways in which the initial global state and the subsequent global states can be perturbed. This approach is traditionally called reachability analysis or perturbation analysis, and was first recognized by West and Zafiropulo [WZ78, Wes78] as a simple and easy-to-automate technique for the verification of communication protocols.

**Definition 2.8**

Let $G$ and $H$ be global states of a protocol $[] = (\{P_i \mid i \notin I\}, L)$. $G \triangleleft H$ iff $t \in X_j(G)$, for some $i, j \in I$, such that one of the following two conditions holds:

- $t = (s_i^G, -x, s_j^H)$ and $c_{ij}^G = c_{ij}^H x$, while all other elements of $H$ are the same as those of $G$;
- $t = (s_i^G, +y, s_j^H)$ and $c_{ij}^G = yc_{ij}^H$, while all other elements of $H$ are the same as those of $G$.

$H$ is the successor of $G$ by $t$, also denoted as $G \xrightarrow{t} H$. 

The relation $\triangleleft$ is a binary relation over the global states of a protocol. Informally, $G \triangleleft H$ represents a perturbation of a global state $G$ resulting in another global state $H$ via the execution of a single transition of one of the processes, while the other processes and the unaffected simplex channels remain unchanged. This notion is naturally extended to sequences of transitions.

**Definition 2.9**

Let $G$ and $H$ be global states of a protocol $[]$, and denote by $\triangleleft^*$ the reflexive and transitive closure of $\triangleleft$. $H$ is reachable from $G$ iff $G \triangleleft^* H$. When $G = G_0$, $H$ is said to be reachable. The set of all reachable global states of $[]$ is denoted by $R_[]$. For a sequence of transitions $[] = t_1 t_2 \ldots t_m$, $G \triangleleft^* H$ denotes the existence of global states $Q^0, \ldots, Q^m$ such that $G = Q^0 \quad Q^1 \quad \ldots \quad Q^m = H$.

Definition 2.9 formalizes the conventional reachability analysis, yielding the set of all reachable global states of a protocol and the transitions between them. This is referred to as the reachabled global state space of the protocol, which can be viewed as a (possibly infinite) directed graph with nodes and edges corresponding to reachable global states and global state transitions, respectively.

**Definition 2.10**

The reachability graph of a protocol $[]$ is the labeled directed graph with vertex set $R_[]$ such that there is an edge labeled $t$ from $G \in R_[]$ to $H \in R_[]$ iff $G \xrightarrow{t} H$.

In practice, the reachable global state space of a protocol is computed by performing a systematic search of all the global states that are reachable from the initial global state. Figure 2.1 gives a standard algorithm for such a search [Hol91]. This algorithm recursively explores all successor states of all global states encountered during the search, starting from the initial global
state, by executing all executable transitions at these states. It uses a work set \( W \) of global states to be analyzed, and a set \( A \) of global states already analyzed. As pointed out in [Hol91], the order of retrieval of states from set \( W \) is consequential: a depth-first search is performed if \( W \) is a stack, and a breadth-first search is performed if \( W \) is a queue. A brief discussion on the particular benefits of each search strategy, as related to protocol verification, is deferred until Section 2.4. Clearly, the algorithm terminates only if the number of reachable global states is finite. Upon termination, set \( A \) should contain exactly the reachable global states of the protocol. It is not difficult to prove that this is indeed the case [AHU74].

```c
/* A is the set of global states that have been analyzed. */
/* \( W \) is the set of global states that still need to be analyzed. */

/* Initialize: */
A = \Ø
\( W = \{G^0\} \)

/* Loop: */
while \( W \neq \Ø \) do {
    remove an element \( G \) from \( W \)
    add \( G \) to \( A \)
    for all \( t \) in \( X(G) \) do {
        derive the successor \( H \) of \( G \) by \( t \) /* execution of transition \( t \) */
        if \( H \) is NOT already in \( A \) or \( W \) then add \( H \) to \( W \)
    }
}
```

**Figure 2.1** Standard perturbation algorithm.

### 2.3 Example: a simple network access protocol

As an example of a communication protocol, consider a system in which a client process seeks access to a network through communication with a server process [ZW+80]. This simple network access protocol can be specified in the CFSM model as follows:

- \( P = \{P_1, P_2\} \), where \( P_i = (S_i, s_i^0, M_i, D_i) \) \((i = 1, 2)\) with
  - \( S_1 = \{10, 11, 12\}, S_2 = \{20, 21, 22\} \);
  - \( s_1^0 = 10, s_2^0 = 20 \);
  - \( M_1 = M_2 = M_{12} \sqcap M_{21} = \{\text{AReq, ATer}\} \sqcap \{\text{APer, ARej}\} \);
  - \( \sqcap_1 = \sqcap_{12} = \{t_1^1, t_1^2, t_1^3, t_1^4\}, \sqcap_2 = \sqcap_{21} = \{t_2^1, t_2^2, t_2^3, t_2^4\} \), where
Chapter 2  Protocol design and analysis: the CFSM model

\[ t_1^1 = \{10, -AReq, 11\} \quad t_1^2 = \{20, +AReq, 21\} \quad (Access \ Request) \]
\[ t_1^3 = \{11, +ARej, 12\} \quad t_1^4 = \{21, -ARej, 22\} \quad (Access \ Rejected) \]
\[ t_2^1 = \{11, +APer, 12\} \quad t_2^2 = \{21, -APer, 22\} \quad (Access \ Permitted) \]
\[ t_2^3 = \{12, -ATer, 10\} \quad t_2^4 = \{22, +ATer, 20\} \quad (Access \ Terminated) \]

\[ L = \{\{1, 2\}, \{2, 1\}\} \text{ (i.e. there are two error-free simplex channels } C_{12} \text{ and } C_{21}\).

\[ C_{12} \]

\[ C_{21} \]

\[ \begin{array}{c}
\text{P}_1 \text{ (Client Process)} \\
\begin{array}{c}
10 \\
11 \\
12 \\
\end{array} \\
\downarrow \quad \downarrow \quad \downarrow \\
+\text{ARej} \quad -\text{AReq} \quad -\text{ATer} \\
\downarrow \quad \downarrow \\
+\text{APer} \\
\end{array} \quad \begin{array}{c}
\text{P}_2 \text{ (Server Process)} \\
\begin{array}{c}
20 \\
21 \\
22 \\
\end{array} \\
\downarrow \quad \downarrow \\
+\text{ARej} \quad +\text{ATer} \\
\downarrow \quad \downarrow \\
-\text{APer} \\
\end{array} \]

\[ C_{12} \quad \cdots \quad C_{21} \]

\[ \begin{array}{c}
\text{Figure 2.2} \text{ A network access protocol.} \\
\end{array} \]

Process \( P_1 \) models the client process and process \( P_2 \) models the server process. The respective process graphs are shown in Figure 2.2. Initially, each process is at its initial state (process states 10 and 20, respectively) and both simplex channels are empty. The only event that can occur in this initial global state is the transmission of an Access Request message by \( P_1 \) (transition \( t_1^1 \)). This event causes \( P_1 \) to move from state 10 to state 11 and the message “AReq” to be placed in channel \( C_{12} \). At this point \( P_1 \) cannot progress since only receive transitions are specified at state 10 while channel \( C_{21} \) is still empty. With “AReq” in \( C_{12} \), however, process \( P_2 \) can receive this message by executing transition \( t_2^1 \). In doing so it moves from state 20 to state 21, and channel \( C_{12} \) becomes empty again. Note that there is no assumption on the time a message spends in a channel, i.e. the delay between the transmission and the reception of a message is variable and unspecified. \( P_2 \) can now continue by putting either “ARej” or “APer” in \( C_{21} \) (transition \( t_2^2 \) or \( t_2^3 \)), corresponding to rejecting or permitting network access, respectively. In either case, \( P_2 \) reaches a state at which it cannot execute any transition until \( P_1 \) places a new message in channel \( C_{12} \). A further step-wise analysis of the joint behavior of the two processes readily yields the complete reachable global state space of the protocol, as shown by the reachability graph in Figure 2.3. It contains 8 global states.
and 10 transitions. An inspection of this graph quickly reveals that the contents of both simplex channels remain finite, $C_{12}$ holding at most two messages and $C_{21}$ at most one message. Also, the protocol seems to manifest a “healthy” cyclic behavior since it can eventually return to its initial global state.

Figure 2.3 The reachability graph of the network access protocol in Figure 2.2.

### 2.4 Protocol verification by reachability analysis

For the network access protocol above, a simple “walkthrough” by simulating the processes provides considerable insight of the behavior of this protocol. Inspecting the reachability graph is easy because it is small. One must realize, however, that the 8 reachable global states emerge from a potentially much larger number. Let us assume that simplex channel $C_{12}$ holds at most two messages from the set \{AReq, ATer\} and that simplex channel $C_{21}$ holds at most one message from the set \{ARej, APer\}. With processes $P_1$ and $P_2$ each having 3 process states, there are in fact $3 \times 3 \times (2^0 + 2^1 + 2^2) \times (2^0 + 2^1) = 189$ syntactically distinct global states. Although 181 of these states play no role in the behavior of the protocol (as they are not reachable from the initial global state), this calculation demonstrates the potential exponential growth of states that may result from
reachability analysis, even when assuming a very limited number of messages in the channels at any
time. As a consequence, analyzing the behavior of a protocol is generally much more complex. It
requires precise definitions of error conditions or correctness conditions in order to determine the
existence of design errors by examining the reachability graph constructed during reachability
analysis.

The correctness conditions commonly considered for protocols specified in the CFSM model
are defined in terms of erroneous transitions and (reachable) global states. It is usually desirable
that a protocol be free from non-executable transitions, i.e. transitions that cannot be executed at
any reachable global state. A non-executable transition compares to “dead code” in a computer
program and indicates a design error. Presumably, the protocol designer has included the transition
believing it would be used. The design needs to be reconsidered: the transition is either removed,
with no effect on the behavior of the protocol, or the transition is made executable by adding or
modifying other transitions.

Definition 2.11
Let \( \square = (\{P_i\} \mid i \in I), L \) be a protocol and \( t \square \bigcup_{i \in I} L_i \) a transition. \( t \) is non-executable iff \( \square G \square R_{ij} \exists: t \square X(G) \).

A protocol should generally also be free from deadlocks. A deadlock is a reachable global state in
which all channels are empty and none of the processes is ready to send a message. This is a
special case of a non-progress state, in which the channels need not be empty but no process is
able to execute a transition. A non-progress state normally testifies that the protocol has come to an
“unexpected halt”. The unexpected nature of halting is what indicates a design error. A deadlock
state can also be seen as a special case of a stable state. Stable states are reachable global states with
all channels empty and irrespective of the ability of processes to execute transitions. They can be
useful for detecting loss of synchronization [Wes78].

Definition 2.12
Let \( \square = (P, L) \) be a protocol. A global state \( G \square R_{ij} \) is a progress state iff \( X(G) \neq \emptyset \); otherwise, \( G \) is
a non-progress state. \( G \) is a stable state iff \( \square (i, j) \mid L \): \( c^G_{ij} = \emptyset \) \( G \) is a deadlock state iff it is a non-
progress state and a stable state. \( \square \) progresses indefinitely iff \( \square G \square R_{ij} \exists: G \) is a progress state.

Finally, a protocol should generally not exhibit any unspecified reception states. An unspecified
reception state refers to a reachable global state in which a message at the front of some queue
cannot be received due to a missing (i.e. unspecified) receive transition. The design error here is a
case of “underspecification”, in contrast to a non-executable transition which signifies
“overspecification”.

Definition 2.13
Let \( \mathcal{G} = (P, L) \) be a protocol. A global state \( G \mathcal{R} \mathcal{G} \) is an unspecified reception state iff \( \mathcal{G}(j, i) \mathcal{R} L: \text{front}(c_{ij}^G) = y \) and \( (s_{ij}^G, +y, s) \mathcal{R} \mathcal{G} \). The pair \( (s_{ij}^G, y) \) is called an unspecified reception (ur-pair for short) for process \( P_i \) (in \( G \)).

Note that a non-progress state is either a deadlock state or an unspecified reception state. In the latter case one also speaks of a blocking unspecified reception state. The absence of both deadlocks and unspecified receptions thus implies the absence of non-progress states.

The above correctness conditions have been used to verify whether the communication among the processes in a protocol progresses indefinitely [Gou84], and whether the protocol itself is complete and logically consistent [ZW+80]. This explains why they are also referred to as progress properties or logical correctness properties. A protocol is often said to be logically correct when it is free from deadlocks, unspecified receptions and non-executable transitions. Verifying the logical correctness of a protocol by reachability analysis amounts to constructing the entire reachable global state space of the protocol while checking each reachable global state against the respective correctness conditions. The verification algorithm is thus the same as the standard perturbation algorithm in Figure 2.1, apart from a few straightforward additions to report any violations of logical correctness properties. As mentioned in Section 2.2, this algorithm allows a breadth-first search as well as a depth-first search of the reachable global state space. The former has the advantage of exposing the shortest path to an “error state” first. The latter has the advantage that it does not require extra information to be stored in order to reconstruct a path to such a state: a path is implicitly defined by the content of the stack. Also, as the depth of a search tree is usually much smaller than its breadth, a depth-first search often requires considerably less memory to maintain the set \( W \) of states to be analyzed [Hol91]. Saving memory in case of a depth-first search is further possible by storing only the global states in the “current” search path. The set \( A \) of analyzed states is then not needed. Of course, this is generally at the expense of running time for it can no longer be determined whether a newly generated global state has already been encountered in a previous search path. Redundant explorations of global states are thus likely to be carried out.

Example 2.14
Consider again the network access protocol in Section 2.3. It follows from the reachability graph in Figure 2.3 that this protocol is logically correct. Indeed, there are no deadlock states since all the reachable global states have an “outgoing” transition. Furthermore, all transitions are executable since each transition appears at least once in the reachability graph. Finally, there are no unspecified reception states since all the reachable global states which still contain a message at the front of an incoming queue have an “outgoing” receive transition for that message.
Example 2.15
Consider the protocol depicted by Figure 2.4. Figure 2.5 illustrates part of its reachability graph which consists of a total of 25 global states and 35 transitions. One can see that the protocol contains at least one deadlock state and three unspecified reception states (one being a non-progress state as well). By completing the reachability graph, one can further see that the three transitions (22, \(+a\), 23), (23, \(-d\), 22) and (11, \(+d\), 10) are non-executable transitions.

In principle, proving a protocol correct requires more than just verifying the absence of deadlocks, unspecified receptions and non-executable transitions. A protocol may be logically correct and yet not perform its intended functions. The traditional focus of reachability analysis is nevertheless on general properties that are of interest to all protocols independent of their intended functionality [BZ83, ZW*80], such as the ones defined. The differentiation between general correctness requirements and functional, protocol-specific requirements appears natural and has long been accepted. This is witnessed for example by the following citation, hinting at an analogy between the syntactic correctness of a program and the logical correctness of a protocol [Rud88]:

> “Just as successfully passing through a syntax-checking precompiler is no guarantee that a program written in a high-level language will perform its intended function, validation of a protocol does not guarantee that the protocol will perform its intended function.”

Here, “validation” actually means “verification” for the objective is to prove a protocol logically
correct. Violations of logical correctness properties are thus interpreted as “syntactic” errors in the protocol, while violations of functional requirements can be seen as “semantic” errors.

Logical correctness properties also classify as so-called safety properties. Safety properties are intuitively characterized as stating that “bad things” never happen, as opposed to liveness properties which state that “good things” do eventually happen [Lam77]. The violation of a logical correctness property (e.g. the occurrence of a deadlock or unspecified reception) then corresponds to a “bad thing” that is irremediable and takes place fundamentally after a finite sequence of execution steps of a protocol. Liveness properties are typically used to claim that a protocol does something useful, like performing its intended functions (e.g. faithful message transfer). Most properties of this kind can be violated by infinite execution sequences only [Sis85, AS87]. We will return in detail to the analysis of general safety and liveness properties in Chapter 7.

2.5 Challenges in reachability analysis

Reachability analysis has become an established technique for protocol verification, and several notable “success stories” about applying this technique to complex, industrial-size protocols have
been reported over the past 15 years (e.g., see [Rud92]). Nevertheless, there are two fundamental problems with reachability analysis which render its use impractical, still, for most real protocols, namely undecidability and state explosion.

2.5.1 Undecidability

In [BZ81], Brand and Zafiropulo showed that most properties of interest, such as logical correctness properties, are in general undecidable for protocols specified in the CFSM model. Undecidability stems from the potential unboundedness of the channels in the model. Even a protocol consisting of two processes communicating over two channels with unbounded capacity appears as powerful as a Turing machine. In essence, the halting problem reduces to the problem of detecting (logical) design errors in a protocol by using the unbounded channels to simulate the tape of a Turing machine [BZ81].

Despite this negative result, decidability of the verification problem is known for some classes of protocols. Most eminently, reachability analysis decides the verification problem for all those protocols whose channels are bounded.

Definition 2.16

Let $\mathcal{P} = (P, L)$ be a protocol and $C_{ij}$ a simplex channel, with $(i, j) \in L$. $C_{ij}$ is bounded iff $\exists K \geq 0 \forall G \in R_{ij}: |c_{ij}^G| \leq K$. □

Definition 2.17

A protocol $\mathcal{P}$ is bounded iff $R_{ij}$ is finite. □

Clearly, from these two definitions a protocol is bounded iff all its simplex channels are bounded. Although boundedness of channels is also undecidable in general, most common protocols do not make use of the full generality of the CFSM model. Brand and Zafiropulo observe [BZ83, p. 329]:

“[…] many practical protocols do have all their channels bounded. Protocols with unbounded channels usually use them in a simple manner, which makes them worth considering.”

Other classes of protocols for which the verification problem is decidable by reachability analysis similarly originate from restricting the allowable sequences of messages that can be in transit at any time. These classes are identified by, for instance, a language-like requirement on the contents of each channel [Pac87, Fin88, Oku88], or by the boundedness of specific channels in combination with a restricted communication topology [BZ83, GH85, PP90, CR93, LM94].

Lastly, for unbounded protocols one can elude undecidability to some degree by prescribing bounds on the channel capacities. A protocol may then not be fully analyzable but it can be verified correct by approximation [BZ83]. Precisely, by assigning to each simplex channel of a protocol an
explicit bound on the maximum number of messages it may hold, and thereby preventing the execution of send transitions involving full channels (cf. Definition 2.6), the reachability graph of the protocol is guaranteed to be finite. Channels with prescribed capacity bounds will be referred to as prebounded channels. For protocols with prebounded channels an additional logical correctness property is of interest, namely freedom of buffer (or channel) overflows [WZ+80].

**Definition 2.18**

Let \( \mathcal{P} = (P, L) \) be a protocol and denote by \( B_{ij} \) the bound on a simplex channel \( C_{ij} \). A global state \( G \in \mathcal{R} \) is a buffer overflow state iff \( (i, j) \in L : |c_{ij}^G| = B_{ij} \) and \( (s_i^G, -x, s) \in D_{ij} \), for some \( x \in M_{ij} \). The pair \( (s_i^G, x) \) is called a buffer overflow (bo-pair for short) for process \( P_i \) in \( G \).

Buffer overflow states are thus reachable global states in which some process is ready to send a message onto a full channel. Detecting the underlying bo-pairs is beneficial for several reasons. A bo-pair may indicate that the prescribed bound on a channel is not sufficient and “restricts” the behavior of a protocol in an adverse way or, conversely, it may reveal that the protocol is unbounded which is likely to be a design error as well. Also, a bo-pair may indicate that a channel will overflow in practice in case of a conforming protocol implementation where processes do not know the status of their outgoing buffers. This would typically result in oblivious message loss.

### 2.5.2 State explosion

When a protocol is bounded it is theoretically possible to explore and analyze its entire reachable global state space. In practice, however, even the state space of a bounded protocol may be restrictively large for exhaustive search. As illustrated at the beginning of Section 2.4, simple combinatorics affirm that the number of reachable global states may grow exponentially, a phenomenon known as the state explosion problem. This problem comes into effect when the size of the state space surpasses the amount of memory available for the search: exhaustive state exploration becomes impractical.

Fortunately, the state explosion problem is not always unavoidable. It is well-known that many protocols manifest a large number of reachable global states and transitions that are redundant for verification purposes. Indeed, for nearly two decades much effort has been spent on devising techniques that exploit this redundancy and thereby relieve the state explosion problem. Such relief strategies enable the verification of properties of protocols by examining only part of their state spaces. The next chapter provides an overview of existing relief strategies, in particular of those pertaining to the CFSM model.