Chapter 3

Relief strategies to tackle state explosion

The principle objective of most improved verification techniques is to relieve the intricate problem of state explosion, without compromising too much the ability to verify several different types of correctness properties. This chapter gives an overview of the various relief strategies developed to this extent. In line with the previous chapter, we focus primarily on relief strategies that are based on state exploration. We point out that two excellent surveys of existing relief strategies already appeared in the late 80’s [LCL87, Yua88]. Our overview adopts the structure of [Yua88] and includes also relief strategies that have been proposed more recently.

3.1 Improved state exploration techniques

One of the more prominent causes of the state explosion problem is the modeling of concurrency by interleaving or, precisely, the exploration of all possible interleavings of concurrent transitions. For instance, in reachability analysis the execution of $k$ concurrent transitions is examined by exploring all $k!$ orderings of these transitions. Yet, many properties of interest are insensitive to the order of concurrent transitions. Protocols therefore often manifest a large number of reachable global states and transitions that are redundant for verification purposes. It is this observation which has led to the development of several improved state exploration techniques – techniques that reduce the complexity of conventional reachability analysis by examining just part of the state space of a protocol, a part that is provably sufficient to verify a given property.

3.1.1 Fair reachability analysis

Fair reachability analysis has emerged to date as an improved state exploration technique for the verification of cyclic protocols, in which two or more processes form a unidirectional ring. The technique has been proposed incrementally by several researchers [RW82, GH85, LM94, LM96].
**Canonical sequences**! Fair reachability analysis was first proposed by Rubin & West as a nameless relief strategy for the verification of two-process protocols (i.e. networks of two processes communicating over two error-free simplex channels) [RW82]. They recognized that much of the redundancy in conventional perturbation analysis can be eliminated by imposing an order among transitions when both processes can execute a transition at a global state. It was shown that transitions can then always be executed pair-wise, one of each process, resulting in so-called *canonical sequences*. Instead of executing just a single transition of one process, “matching” transitions of both processes are coupled to generate the next global state. Exploring only canonical sequences can substantially reduce the number of global states and transitions analyzed, as only the states with an equal number of messages in both channels are generated. Rubin & West argued that this improvement in efficiency does not compromise the detection of deadlocks and unspecified receptions in two-process protocols.

**Fair progress state exploration**! Gouda & Han built on the technique above, and actually introduced the name fair reachability analysis [GH85]². This name stems from the recognition that two processes progressing through a canonical sequence factually progress with the same (relative) speed. Put differently, a canonical sequence does not allow one of the processes to execute more transitions than the other at any given time, and is hence fair with respect to their progress speeds. Gouda & Han supported the claims in [RW82], viz. for a two-process protocol the fair reachability graph obtained by “fair progress state exploration” is usually much smaller than the corresponding reachability graph and, when finite, this graph can be used to decide the absence of deadlocks and unspecified receptions. They also learned that boundedness detection, which had not yet been addressed, is unsolvable by fair reachability analysis itself. Consequently, algorithms were presented which *augment* the fair reachability graph of a two-process protocol in order to decide whether the protocol is bounded [GH85]. The algorithms assume a *finite* fair reachability graph, which can then be employed also to find the smallest possible capacities of the two channels in the protocol. These contributions elevated fair reachability analysis as a rather effective relief strategy, suitable for detecting the most common logical errors in two-process protocols.

**Generalized fair reachability analysis**! Recently, Liu & Miller generalized the technique of fair reachability analysis to *n*-process cyclic protocols, where \( n \geq 2 \) processes are joined by \( n \) simplex channels in a closed loop [LM94, LM96]. All channels are oriented in the same direction, hence forming a unidirectional ring. A two-process protocol is thus a special instance of a cyclic protocol.

---

² The name fair reachability analysis was in fact introduced earlier by Yu & Gouda [YG82], who presented an algorithm for deadlock detection which is polynomial in both time and space. However, the protocols considered are assumed to have two processes that can only send and receive one type of message, while no process state may involve both send and receive transitions. These assumptions are too restrictive in practice.
The key aspect of the generalization consists in preserving the *equal channel length property*: each global state generated by fair reachability analysis is a reachable global state in which all channels hold the same number of messages. As pointed out above, this property was already identified implicitly in [RW82, GH85]. Indeed, for two-process protocols the equal channel length property coincides with the perception of equal progress speed.

Liu & Miller showed that the decidability results established for two-process protocols are valid for all cyclic protocols in general. That is, the detection of deadlocks and unspecified receptions, and boundedness detection are decidable for any cyclic protocol whose fair reachability graph is finite. In addition, they determined the decidability of detecting non-executable transitions for these protocols, which had not been established before for two-process protocols. The fair reachability graph of a cyclic protocol was found to be finite if (at least) one of the channels in the protocol is bounded. This also follows the same result achieved earlier for two-process protocols [GCL85]. A necessary and sufficient condition for finiteness was identified as well, namely the absence of indefinite simultaneous growth of all channels in a cyclic protocol. This condition thus completely characterizes the subclass of cyclic protocols for which the detection of deadlocks, non-executable transitions, unspecified receptions and unboundedness is decidable [LM94, LM96].

Besides being an effective and efficient relief strategy for cyclic protocols, an important extra asset of fair reachability analysis is that it can solve the verification problem for various *unbounded* protocols. The conventional reachability analysis fails to deal with such protocols due to the infinity of their state spaces, induced by an unbounded accumulation of messages in (one of) the channels (cf. Section 2.5.1). Yet, the number of global states explored by fair reachability analysis may very well be finite. The technique of fair reachability analysis is presented in further detail in Chapter 4, along with one of our own contributions: a generalization of fair reachability analysis to so-called multi-cyclic protocols.

3.1.2 “Reduced” reachability analysis

For two-process protocols, two other improved state exploration techniques have appeared which are quite similar in nature to fair reachability analysis. Zhao & vonBochmann [ZB86] proposed a “reduced” reachability analysis on the basis of a different representation of the CFSM model, in which protocols are modeled by process equations. State exploration is then performed through algebraic transformation rules. In order to reduce the space and time requirements, the proposed method employs both conventional and fair progress schemes in the generation of global states. It was first shown to detect all non-progress states, i.e. all deadlocks and blocking unspecified receptions, and subsequently extended to enable the detection of all unspecified receptions [ZB86].
Cacciari & Rafiq [CR93] presented a reduced reachability analysis for two-process protocols that resembles Zhao & von Bochmann’s technique. They again revert to the plain CFSM model, however, and also incorporate internal transitions in the model. Cacciari & Rafiq claimed that their method improves the one in [ZB86] in the sense that it allows more properties to be verified without increasing the order of magnitude of the number of generated states. The reachability graph constructed by the proposed method spans all reachable global states in which either both channels hold the same number of messages (cf. the equal channel length property), or one of the channels is empty while the other holds one message. This reduced reachability graph thus includes all deadlock states and has further been shown appropriate for verifying the absence of unspecified receptions and certain livelocks (precisely, blocking cycles). Not all unspecified receptions are necessarily detected though, but it is guaranteed that at least one unspecified reception manifests itself in the reduced reachability graph if there exists one in the protocol.

3.1.3 Maximal progress state exploration

Like fair progress state exploration, maximal progress state exploration is a technique that attempts to eliminate redundancy in conventional reachability analysis by not examining all relative progress speeds of processes [GY84]. Its applicability is limited to two-process protocols. Rather than forcing the two processes to progress at equal speed, this technique forces one of the processes to make maximal progress. This means that transitions of one process, say $P_1$, are executed as much as possible, while the other process $P_2$ remains inactive. Process $P_2$ comes into play only when $P_1$ can no longer progress, i.e. when all transitions at its current process state are receive transitions that cannot be executed. In this case, state exploration continues by executing transitions of $P_2$ until one of the receive transitions of $P_1$ becomes executable. Process $P_1$ then resumes progress and the procedure repeats itself as long as new global states can be generated.

Gouda & Yu proved that all non-progress states are detected by performing maximal progress state exploration for either process. In addition, they proved that all buffer overflows (in case of channels with finite capacity) can be identified by performing maximal progress state exploration for both processes, once for $P_1$ and once for $P_2$. Detecting buffer overflows is thus divided into two independent subtasks, each of which generally requires less space and time than the combined task. The overall time requirements may then be reduced by executing the two subtasks in parallel, at the expense of an extra processor, whereas the space requirements may be reduced by executing the two subtasks in sequence on a single processor [GY84].

---

3 Although considered in [CR93], internal transitions do usually not have a significant effect on state exploration based verification techniques. They are therefore largely ignored by researchers in the field.
3.1.4 Reduced implementation sequences

The relief strategy proposed by Itoh & Ichikawa [II83] (see also [KI+85]) is applicable to protocols with any number of processes (at least two, of course). Constraints are imposed, however, on the structures of these processes: all processes must synchronize on their initial process states after a finite number of execution steps and no process is allowed to have a cyclic execution that does not pass through its initial state. Although these constraints ensure that the global state space of a protocol is finite, eluding any such embedded cycle in the process graph of a process is surely restrictive in practice. Even a simple data transfer protocol usually exhibits at least one embedded cycle (e.g. the retransmission part of the sender in an alternating bit protocol).

Itoh & Ichikawa’s technique entails the simultaneous (or parallel) execution of transitions of different processes in a global state to derive the next global state. Intuitively, the aim is to abstain as much as possible from any execution order among concurrent transitions. Special attention is thereby required for transitions that are not executable at the current global state, say $G$, but that may still become executable later at a global state reachable from $G$. Such transitions are called potentially admissible events. For each global state encountered with potentially admissible events, additional simultaneous progress schemes are considered by inhibiting the execution of transitions of the processes in which these events arise. As shown in [II83], this procedure may result in the analysis of just a small part of the global state space of a protocol. Indeed, the proposed technique examines only the so-called reduced implementation sequences of a protocol, which constitute a subset of all the possible protocol executions. A reduced implementation sequence is an execution from the initial global state of the protocol to either a non-progress state or a global state in which all processes have returned to their initial process states (the channels need not be empty). The set of reduced implementation sequences is used to verify the protocol against a given requirement specification, viz. a set of prescribed protocol executions. Strictly speaking, the intent of Itoh & Ichikawa is thus not to verify logical correctness properties, but rather “operational” requirements of a protocol [II83]. Regarding the former, their technique does not lend itself for detecting logical errors other than non-progress states.

3.1.5 Simultaneous reachability analysis

Simultaneous reachability analysis is a state exploration technique which generalizes the ideas behind the above work by Itoh & Ichikawa. It was proposed by Özdemir & Ural [ÖU94, ÖU95, Özd95] as a relief strategy for verifying logical correctness properties of protocols with an arbitrary number of processes, arbitrary communication topology and arbitrary process structures. That is, in contrast to the technique in [II83], restrictions no longer apply to any of the protocol attributes.
Simultaneous reachability analysis was shown to detect all non-progress states and non-executable transitions of a protocol. Furthermore, augmentations were devised to enable the detection of all unspecified receptions and all overflowed channels. An overflowed channel refers to a simplex channel for which there exists a bo-pair (cf. Definition 2.18). Note that the detection of all bo-pairs implies the detection of all overflowed channels, but not vice versa. We will return in more detail to simultaneous reachability analysis in Chapter 5, where we propose an incremental improvement of this relief strategy.

3.1.6 Partial-order reduction methods

Partial-order reduction methods [God90, Val90, HGP92, KP92a, Val92, Val93, GW93, GW94, HP95, Pel96] are a collection of cognate techniques to alleviate the state explosion problem in verifying finite-state concurrent systems (including communication protocols). Although these techniques have been proposed predominantly for systems modeled as Petri Nets (see e.g. [Pet81]), and for systems defined with CSP/CCS-style semantics [Hoa85, Mil89], they apply in principle to all models that express concurrency by interleaving [HP95, God96]. Partial-order reduction methods are effective and generally efficient for verifying local and termination properties (e.g. freedom of non-progress states and non-executable transitions [God90, Val90, HGP92, KP92a, GW93] and, moreover, for verifying linear-time temporal logic (LTL) properties [Val92, Val93, GW94, HP95, Pel96]. The latter is known as LTL model-checking, and captures arbitrary (temporal) safety and liveness properties of concurrent systems [Lam77, Pnu77, Lam80, Lam83, WVS83, LP85, VW86, AS87, Wol89, MP92].

Like the improved state exploration techniques discussed above (and specific to the CFSM model), partial-order reduction methods exploit the fact that in many cases the properties verified are insensitive to the order of concurrent transitions. They aim at exploring just one fixed order among concurrent transitions at global states, by executing at each global state encountered during state exploration only a discriminating subset of the transitions executable at that state, rather than all of them. This then yields a reduced state space which is guaranteed to preserve the property under consideration. Partial-order reduction methods are also discussed in more detail later in the thesis. Chapter 7 presents an approach of our own to LTL model-checking, which we will propose as an enhancement of the partial-order approach.

3.2 Closed covers

The “closed-cover” technique by Mohamed Gouda [Gou84] is somewhat of an intruder compared to most other relief strategies in the sense that its sole objective is to prove the absence of protocol
design errors rather than showing their existence. We quote [Yua88, p. 167]:

“this technique is a theoretical school of thought which believes that proving protocols free from errors is much more significant than detecting errors, and detecting no errors is not sufficient enough to show protocols are free from errors [Gou84].”

A closed cover is basically a set of global states of a protocol containing only the initial global state and global states that are someway “closed” with respect to reachability (we refer to [Gou84] for a precise definition). It was proven that the existence of a closed cover is sufficient (and in many cases necessary) to guarantee indefinite progress for a two-process protocol. The technique thus consists in finding a closed cover which, unfortunately, must be “guessed” and is hence a difficult task. Another drawback of the closed-cover technique is its inability to verify the possibility of non-progress, but then again this complies with the aim of proving the absence instead of the presence of errors. Two clear advantages are that the size of a closed cover is usually smaller than the size of the state space of a protocol and that progress of unbounded protocols may be verified as well (cf. fair reachability analysis). Gouda claims that his technique can be extended in a straightforward manner to verify progress for protocols with more than two processes. He also signifies an analogy between the closed-cover technique and the assertion techniques to verify safety properties of sequential programs. In this analogy, “closedness” of a global state corresponds to the requirement that each assertion before a block of statements in a sequential program must be sufficient to ensure the assertion after the block.

### 3.3 Acyclic expansions

In [BZ83, KWN88], acyclic expansion techniques were proposed which attempt to overcome the state explosion problem by avoiding altogether the construction of a global reachability graph. Instead of exploring the global states of a protocol as a whole, these techniques consider processes separately by expanding their process graphs into local trees. Global information is then added to the trees such that each local tree represents all possible (global) executions of the corresponding process. Design errors such as deadlocks, unspecified receptions and buffer overflows can be detected during the construction of the local trees.

In comparison with conventional reachability analysis, the acyclic expansion techniques reduce the number of analyzed states from $O(m_1 \Box m_2 \Box \ldots \Box m_n)$ to $O(m_1 + m_2 + \ldots + m_n)$, where $n$ is the number of processes and $m_i$ is the number of process states of process $P_i$. Thus, when a protocol is rather complex, involving a large number of process states per process, these techniques clearly outperform reachability analysis. An additional advantage is the modularity: processes can easily be modified without affecting the complete analysis, as all local trees are maintained individually. A considerable drawback of the approach is the algorithmic complexity. Functions are used to acquire
and add global information to the local trees. Their implementation requires complex tree searching procedures, which make the overall verification algorithm much more complicated than the standard perturbation algorithm.

### 3.4 Divide-and-conquer strategies

The following techniques follow the well-established divide-and-conquer paradigm to problem solving. A protocol is decomposed or partitioned into components which are subsequently verified separately to ensure the correctness of the protocol itself. The complexity of verification is thus relieved since the components are usually smaller in the number of states and transitions than the original protocol.

#### 3.4.1 Duologue-matrix analysis

Duologue-matrix analysis is one of the first automated protocol validation techniques [Zaf78]. The technique was proposed for two-process protocols only, where each process must further obey the condition that any cycle in its process graph passes through the initial (or some quiescent) process state. Recall that this restriction was also in effect for the technique in [II83] based on reduced implementation sequences, which can then be seen as an extension from the two-process model here to multiple processes.

Duologue-matrix analysis starts off by decomposing each process into paths, or *unilogues*, that begin and end in the initial process state. The sets of unilogues of both processes are coupled to form *duologues* (i.e. sequences of two-process interactions) and the duologues are represented in a *duologue matrix*. This matrix basically resembles the Cartesian product of the sets of unilogues. Subsequently, each duologue is classified as being either well-behaved, non-occurrenceable or erroneous, and assigned the value +1, 0 or −1, respectively. A duologue is well-behaved if it always returns to the initial global state of the protocol and if all messages sent along one unilogue are received along the other. Clearly, this prohibits the occurrence of deadlocks and many unspecified receptions. A duologue is non-occurrenceable if it can never be executed or if its execution always results in the execution of another duologue. These duologues have thus little effect on the actual protocol behavior. Lastly, a duologue is erroneous if it is neither well-behaved nor non-occurrenceable. Each duologue in the duologue matrix is then replaced by its corresponding value, yielding a *validation matrix* that can be used to detect design errors. In particular, a protocol contains an error if its validation matrix contains any −1 elements, and the positions of these elements identify the erroneous duologues of the protocol. Furthermore, if one of the rows or columns of the validation matrix contains more than one +1 element, then the behavior of one process is ambiguous with
respect to the other. It turns out that the validation matrix of a “perfect” protocol is an identity matrix. Duologue-matrix analysis has proven to be useful for the detection of design errors in real protocols, such as the X.21 recommendation of CCITT [WZ78].

### 3.4.2 Decomposition methods

Vuong & Cowan observed that large, well-designed protocols represented by finite directed graphs often exhibit some basic structures which allow them to be decomposed into several smaller component graphs [VC82]. These components can be verified separately to yield a verdict about the original protocol. Three basic structures are identified (viz. the nested, sequential and parallel structures), each of which enables a specific decomposition scheme. The decompositions turn out rather simple and powerful for protocols which indeed demonstrate these structures, but are suited for two-process protocols only. Similar methods were proposed in [CM83, CM86, CGL85], where also protocols with irregular structures are partitioned into components (or multi-phases [CGL85]).

### 3.4.3 Protocol projections

Another decomposition approach is based on the idea of protocol projections and aims at protocols with several distinguishable functions [LS84]. A protocol undergoes a functional decomposition by means of projections. The extracted functions are verified individually on the basis of so-called image protocols. An image protocol is constructed for each function by aggregating groups of states, messages and events of the corresponding entities in the original protocol. Image protocols are typically smaller and easier to analyze.

### 3.5 Partial state exploration

Partial state exploration techniques are motivated by the perception that analyzing just that part of the state space with the largest probability of occurring may be “good enough” to judge a protocol correct. Such techniques may provide a good alternative if the amount of available memory is insufficient for exhaustive analysis [Hol91], or even for improved state exploration techniques (cf. Section 3.1). The inherent drawback of partial state exploration is, of course, that it cannot be used to verify the absence of errors (or conversely, the error coverage attained cannot be measured).

### 3.5.1 Random-walk state exploration

Colin West [Wes86] proposed a variation of conventional reachability analysis, in which a new global state is derived at random by selecting arbitrarily one transition to be executed at the current
3.5.2 Probabilistic verification

Instead of performing exhaustive state exploration, a probabilistic verification approach examines the “most probable” execution sequences of a protocol. It operates under the assumption that the probability of encountering a state or transition with low probability of occurrence is very small in a real execution sequence of a protocol. Protocols with errors in sequences that are not likely to be executed are then considered to be acceptable. A concrete probabilistic verification technique is given in [MS87]. As pointed out in [Rud88, Hol90], the main difficulty with such techniques is estimating the probabilities of occurrence of states and transitions.

3.5.3 Heuristic verification

Heuristic-based verification techniques have been proposed in [Hol87, LCL87]. Heuristics are used to guide the generation of global states in order to detect design errors quickly and more efficiently without incurring too much overhead. According to [LCL87], heuristic information can be applied mainly at three points during state exploration: (1) in deciding which global state to perturb next, (2)
in deciding which transition to execute next, and (3) in deciding which global states to discard. For instance, for a faster detection of deadlocks and unspecified receptions it appears beneficial to prioritize the execution of receive transitions over send transitions, while for buffer overflows it is the other way around. Similarly, respective preference should be given to the perturbation of the global states with the largest number of executable receive transitions, or states with the “longest” buffers. In order to decide which states to discard one can use occurrence probabilities as in probabilistic verification methods. Other typical heuristic suggestions are to minimize a protocol design before verification in terms of the size of its state machines (i.e. the number of process states and transitions) and the amount of concurrency (i.e. “tightly-coupled” systems), and to limit the size of buffers (cf. Section 2.5.1).

### 3.6 Memory management techniques

The supertrace or bit-state hashing technique of Holzmann [Hol88, Hol90, Hol91], described above, is in fact a pure memory management technique. It does not actually reduce the number of global states generated during state exploration, as do the improved state exploration techniques in Section 3.1, but it merely decreases the memory used to store each individual state. The supertrace technique can therefore be used in conjunction with any state exploration technique (i.e. improved or not).

Another memory management technique that can be combined with state exploration techniques is state space caching [Hol85, Hol87, JJ91, GHP92]. It operates in the context of a depth-first search (DFS) strategy. State space caching amounts to storing all the global states in the currently explored execution path of a protocol, i.e. all the states in the current DFS stack, plus as many other global states as possible given the remaining amount of available memory. A limited cache is thus created consisting of selected states that have already been generated. Initially, every global state generated is stored in the cache. When the cache fills up, old states that are no longer in the DFS stack are removed from the cache to accommodate new ones. This technique never attempts to store more states than possible in the cache. Hence, if the size of the cache is larger than the length of the longest execution path of the protocol (i.e. the maximal size of the DFS stack during state exploration), the state space of the protocol will be fully explored even when the state space itself does not fit in memory (the depth of the state space is usually much smaller than its breadth [Hol91]). If the size of the cache is too small, certain execution paths will be truncated.

The problem of using state space caching is that one can no longer determine whether a newly generated global state has already been encountered in a previously explored execution path (see the discussion in Section 2.4 prior to Example 2.14). Surely, exploring a state each time it is encountered is wasteful. State space caching may thus incur many redundant explorations of global states, yielding a potentially dramatic run-time increase. Indeed, during conventional reachability
analysis, almost every global state of a protocol is typically encountered many times, primarily because all explorations of interleavings of concurrent transitions lead to the same state. The use of state space caching in conjunction with conventional reachability analysis will thus likely cause run-time explosion. Yet, using state space caching in conjunction with relief strategies like fair reachability analysis, simultaneous reachability analysis or partial-order reduction methods (or their enhancements proposed later in this thesis) can be very beneficial. Such improved state exploration techniques reduce not only the number of global states, but also the number of transitions explored. They often avoid most of the nonessential explorations of interleavings of concurrent transitions, and many global states will therefore be encountered only once during state exploration. States that are encountered only once do not need to be stored in memory. Indeed, the mere reason for storing states in memory is to avoid multiple explorations of the same state: when an already generated state is generated again later during state exploration, it is not necessary to regenerate all its successors. Even though it is impossible to anticipate which global states are generated only once (this can be determined only after completing state exploration), if most global states are generated just once, the probability that some state will be regenerated is small. Hence, the risk of redundant work when not storing an already generated state becomes small as well. This enables the combined use of state space caching and improved state exploration techniques without incurring too many redundant explorations of global states. The memory requirements can then strongly decrease without a serious increase of the run-time requirements.

A novel use of bit-state hashing and state space caching for verifying concurrent systems and protocols was recently reported in [MK96]. A method was proposed that uses bit-state hashing in a pre-processing step before verification to compute and store the so-called revisiting degree of each state of a system (similar but not identical to the indegree of a node in a directed graph). Both the pre-processing step and the actual verification of the system are performed by a DFS of the system’s state space. During the first DFS, when a state is generated and its current revisiting degree is zero, the revisiting degree is set to 1 and the revisiting degrees of all successor states are calculated recursively. When a state is generated and its current revisiting degree is greater than zero (i.e. the state is regenerated), the revisiting degree is incremented by 1 and backtracking takes place. (All this applies in fact only to states that do not close a cycle when generated, since such states are already on the DFS stack and can thus easily be identified with no additional space needed [MK96]). During the second DFS, i.e. for the purpose of verification, when a state is generated its revisiting degree (stored in the hash table) is decremented by one. If this yields a revisiting degree of zero, the state is removed from memory. As a result, state space caching can be employed during the DFS without incurring any redundant explorations of global states. The memory requirements can still strongly decrease and at the predetermined expense of only one extra DFS of the state space of the verified system, namely for computing the revisiting degrees of the states. Like bit-state
hashing and “classic” state space caching, the method proposed in [MK96] can be applied to any (improved) state exploration technique. In particular, one can carry out the computation of revisiting degrees of states and the subsequent verification effort on the reduced state space spawned by relief strategies such as fair reachability analysis, simultaneous reachability analysis or partial-order reduction methods (or their enhancements proposed later in this thesis) instead of the full state space of the system spawned by conventional reachability analysis.