Temporal Logic for Stabilizing Systems

Y. Lakhnech[1] and M. Siegel[2]

[1] Christian-Albrechts-Universität zu Kiel
Institut für Informatik und Praktische Mathematik II
Preusserstrasse 1-9, 24105 Kiel, Germany
E-mail: yl@informatik.uni-kiel.d400.de
[2] Weizmann Institute of Science, Rehovot 76100
Dept. of Applied Mathematics and Computer Science
E-mail: mis@wisdom.weizmann.ac.il

Abstract. This paper links two formerly disjoint research areas: temporal logic and stabilization. Temporal logic is a widely acknowledged language for the specification and verification of concurrent systems. Stabilization is a vitally emerging paradigm in fault tolerant distributed computing.
In this paper we give a brief introduction to stabilizing systems and present fair transition systems for their formal description. Then we give a formal definition of stabilization in linear temporal logic and provide a set of temporal proof rules specifically tailored towards the verification of stabilizing systems. By exploiting the semantical characteristics of stabilizing systems the presented proof rules are considerably simpler than the general temporal logic proof rules for program validity, yet we prove their completeness for the class of stabilizing systems.
These proof rules replace the hitherto informal reasoning in the field of stabilization and constitute the basis for machine-supported verification of an important class of distributed algorithms.

1 Introduction

This paper links two formerly disjoint research areas: temporal logic and stabilization. Temporal logic is a widely acknowledged language for the specification and verification of concurrent systems [20, 21]. Stabilization is a concept used in various field of computer science s.a. databases and artificial intelligence [15, 24] but has gained its importance as a vitally emerging paradigm in fault tolerant distributed computing [24].

In this paper we give a brief introduction to stabilizing systems and present *fair transition systems* [20] for their formal description. Then we give a formal definition of stabilization by means of *linear temporal logic* (LTL for short) and provide a set of temporal proof rules specifically tailored towards the verification of stabilizing systems. By exploiting the semantical characteristics of stabilizing systems the presented proof rules are considerably simpler than the general temporal logic proof rules for program validity, yet we prove their completeness for the class of stabilizing systems. These proof rules replace the hitherto informal reasoning in the field of stabilization and constitute the basis for machine-supported verification of an important class of distributed algorithms.

The notion of stabilization has been introduced to the field of distributed computing by Edsger W. Dijkstra. In 1974, he presented several mutual exclusion algorithms for passing a token [12] in a ring topology which all had an astonishing property: irrespectively of the initial number of token in the system, after a finite number of transitions the system converged to a global state where exactly one token was present in the ring and from then on passed between the nodes. Dijkstra called this behavior – convergence towards a set of pre-defined desired states – *stabilization*. Nine years later Leslie Lamport highlighted this so far little-known paper as a milestone in the work on fault-tolerant systems and as the most brilliant published work of Dijkstra [17]. Lamport observed that stabilizing systems show the remarkable property of being capable to automatically recover from transient errors, i.e. errors which do not continue to occur during the period of recovery [24] and to remain correct thereafter. Concurrent systems with multiple tokens converge to a single token; graph markings become colorings; spanning forests become spanning trees; and inconsistent routing tables become consistent again. So the concept of stabilization allows to construct extremely robust fault-tolerant systems [10].

Unfortunately stabilization comes at a high price. Stabilizing algorithms are amongst the most complicated objects studied in the filed of distributed computing. Their intricacy stems from an extremely high degree of parallelism which is inherent to *all* stabilizing systems (see [25] for a detailed discussion) combined with the standard problem in distributed computing that processes, cooperating in a network, have to perform actions taken on account of *local* information in order to accomplish a global objective. A lot of impossibility results, proving that a stabilizing solution for certain tasks and communication topologies cannot exist [14, 18] witness the complexity of stabilization as does the following citation, we quote Dijkstra [12]:

> " For more than a year—at least to my knowledge—it has been an open question whether nontrivial (self-)stabilizing systems could exist."

Most systems with distributed control are not stabilizing, i.e., once erroneously in an illegal state they generally never converge to a legal state again.

Since fault-tolerance is an increasingly important branch of distributed computing there is a well defined need for methods which support the design and verification of fault-tolerant systems based on stabilization, independent of technology, architecture, and application. The state-of-the-art so far are handwritten proofs on the basis of an intuitive understanding of the system –commonly described in pseudo-code – and the desired properties. Undoubtly, handwritten proofs add confidence in the correctness of the algorithms. However, there is common consent that correctness proofs of complicated distributed systems are just as error-prone as the systems themselves [22]. This insight resulted in an increasing interest in *automated* verification which comprise *algorithmic* techniques, known as model checking, e.g., [3, 9], and *deductive* techniques, commonly referred to as theorem proving, e.g. [8, 23]. The involvement of stabilizing systems makes tool support for their verification desirable if not indispensable. The prerequisite for tool-support is a formal basis comprising the following:

1. a formal model to describe stabilizing systems,
2. a formal specification language to describe properties of stabilizing systems, and
3. a formal framework to establish that a certain system obeys some specific properties.

This article provides such a formal basis. We use fair transition systems to describe stabilizing systems and advocate that LTL is an adequate specification language for this class of systems. The combination of temporal logic specifications and transition systems allows for algorithmic verification in the finite state case. To cover those cases where algorithmic verification is not applicable we investigate a temporal proof system for stabilizing systems.

These proof rules are considerably simpler than the general proof rules for linear temporal logic [19, 20, 21]. Nevertheless, we prove that the simple rules are just as complete as the general rules when considering stabilizing systems.

This article is organized as follows: In Section 2 we present fair transition systems and linear temporal logic. A brief introduction to stabilization as well as a formal definition by means of LTL is given in Section 3. The framework for formal reasoning about stabilization, consisting of a set of proof rules, is presented in Section 4. Some conclusions and prospects are given in Section 5.

Remark: Due to space limitations we only sketch some of the proofs. All proofs and technical details can be found in [25].

2 Preliminaries

2.1 Fair Transition Systems

For the description of stabilizing systems, formalisms with all kinds of communication mechanisms (shared variables, synchronous and asynchronous communication) are used in the literature, and the execution models vary from pure interleaving to true parallelism. Therefore we use a generic abstract model, namely fair transition systems [20], to formally capture stabilizing systems and their semantical characteristics. This choice guarantees broad applicability of our formalization as well as tool support since most existing analysis and verification tools are based on transition systems.

For the following presentation we assume a countable set Var of typed variables in which each variable is associated with a domain describing the possible values of that variable. A *state* s is a partial function $s : Var \longrightarrow Val$ assigning type-consistent values to a subset of variables in Var. By V_s we refer to the domain of s, i.e. the set of variables evaluated by s. The set of all states interpreting variables in $V \subseteq Var$ is denoted by Σ_V.

Definition 1. A *fair transition system* $A = (V, \Theta, T, WF, SF)$ consists of a finite set $V \subseteq Var$ of state variables, an assertion Θ characterizing *initial states*, a finite set $T = \{t_1, \ldots, t_n\}$ of *transitions* as well as *weak* and *strong fairness constraints* expressed by set $WF \subseteq T$ resp. $SF \subseteq T$.

□

To refer to the components of a fair transition system (fts for short) A we use A as index; the state space of A is denoted by Σ_A and defined by $\Sigma_A \stackrel{\text{def}}{=} \Sigma_{V_A}$. The set of all fts's A with $V_A \subseteq Var$ is denoted by $\mathcal{S}$.

Each transition $t \in T$ is represented by a first order assertions $\rho_t(V, V')$, called a *transition predicate* [20]. A transition predicate relates the values of the variables V_s in a state s to their values in a successor state s' obtained by applying the transition to s. It does so by using two disjoint copies of set V_s. The occurrence of variable $v \in V_s$ refers to its value in state s while an occurrence of v' refers to the value of v in s'. Assuming that $\rho_t(V_s, V_s')$ is the transition predicate of transition t, state s' is called a *t-successor* of s iff $(s, s') \models \rho_t(V_s, V_s')$. A transition t is *enabled* in state s if there exists s' with $(s, s') \models \rho_t(V, V')$; otherwise it is *disabled*. Enabledness of a transition can be expressed by $en(t) \stackrel{\text{def}}{=} \exists V'.\rho_t(V, V')$. We require that T contains the *idling* transition t_{idle} whose transition relation is $\rho_{t_{idle}} : (V = V')$.

We use a standard linear semantics for fair transition systems [19].

Definition 2. A *computation* of an fts $A = (V, \Theta, T, WF, SF)$ is an infinite sequence $\sigma = \langle s_0, s_1, s_2, \ldots \rangle$ of states $s_i \in \Sigma_A$, s.t.

1. $s_0 \models \Theta$,
2. for all $i \in \mathbb{N}$ state s_{i+1} is a t-successor of s_i for some transition $t \in T$,
3. if transition $t \in WF$ is continuously enabled from some point onwards in σ, there are infinitely many $i \in \mathbb{N}$ with $(s_i, s_{i+1}) \models \rho_t(V, V')$,
4. if transition $t \in SF$ is infinitely often enabled in σ, there are infinitely many $i \in \mathbb{N}$ with $(s_i, s_{i+1}) \models \rho_t(V, V')$.

□

The set of computations generated by an fts A is denoted by $[\![A]\!]$. A finite sequence of states that satisfies the conditions 1. and 2. above is called *computation-prefix*.

For a given computation $\sigma = \langle s_0, s_1, s_2, \ldots \rangle$ and $i \in \mathbb{N}$ we define the prefix of σ up to index i by $\sigma_{\leq i} \stackrel{\text{def}}{=} \langle s_0, s_1, \ldots, s_i \rangle$. The suffix of σ starting at index i is the infinite sequence $\sigma_{\geq i} \stackrel{\text{def}}{=} \langle s_i, s_{i+1}, \ldots \rangle$. The set of computation *prefixes* of A is denoted by $\mathit{Pref}(A)$ the set of computation *suffixes* by $\mathit{Suff}(A)$. By σ_i we refer to the $(i+1)$-st state of computation σ.

We use $\{p\}\ t\ \{q\}$ as abbreviation of $(p \wedge \rho_t(V, V')) \to q'$, where q' results from q by replacing all variables of q by their primed version. For finite sets $T = \{t_1, \ldots, t_n\}$ of transitions we define $\{p\}\ T\ \{q\} \stackrel{\text{def}}{=} \bigwedge_{i=1}^{n} \{p\}\ t\ \{q\}$.

Finally, notice that each fts A is *machine closed* [1], that is, each computation-prefix of A is a prefix of a computation of A.

2.2 Linear Temporal Logic

As specification language we use a future fragment of Linear Temporal Logic [20] without next-operator, referred to as LTL$^-$. Formulas are constructed from state formulas of some first order language $\mathcal{L}$, the temporal until operator U, and boolean operators $\wedge, \neg$ applied to formulas. Temporal formulas are denoted by $\varphi, \psi, \ldots$ and state formulas by $p, q, \ldots$. Temporal formulas are interpreted over infinite sequences of states [20].

Definition 3. Given an infinite sequence $\sigma = \langle s_0, s_1, s_2, \ldots \rangle$ of states and $\varphi \in LTL^-$. We define that σ *satisfies* φ, denoted by $\sigma \models \varphi$, as:

$$
\begin{array}{lll}
\sigma \models p & \text{iff} & \sigma_0 \models p, \\
\sigma \models \varphi \wedge \psi & \text{iff} & \sigma \models \varphi \;\text{ and }\; \sigma \models \psi, \\
\sigma \models \neg\varphi & \text{iff} & \sigma \not\models \varphi, \\
\sigma \models \varphi U \psi & \text{iff} & \exists i \geq 0.\ \sigma_{\geq i} \models \psi \;\text{ and }\; \forall j < i.\ \sigma_{\geq j} \models \varphi.
\end{array}
$$

□

We use the standard abbreviations $\Diamond\varphi \stackrel{\text{def}}{=} \mathit{true} U \varphi$ (eventually), $\Box\varphi \stackrel{\text{def}}{=} \neg\Diamond\neg\varphi$ (always) and $\varphi \Rightarrow \psi \stackrel{\text{def}}{=} \Box(\varphi \to \psi)$ (entails). An fts A satisfies a formula $\varphi \in$ LTL$^-$, denoted by $A \models \varphi$, if all its computations satisfy φ. As notation for sequences we use $\circ$ to denote concatenation of sequences $\mathit{last}(\mathit{seq})$ to return the last element of a (finite) sequence seq.

3 A Brief Introduction to Stabilization

Since stabilization has gained its importance as a paradigm for fault-tolerant computing, we address stabilization from this perspective.

3.1 Stabilization for Fault-Tolerance

Stabilization is besides *masking* and *detection/recovery* mechanisms the major paradigm in fault-tolerant computing [4]. Stabilization is studied as an approach to cope with the *effect of arbitrary faults*, as long as some indispensable prerequisites for recovery, characterized by a so-called *fault-span*, are not violated. Such prerequisites may require that the actual code is not affected, or that the communication topology of the distributed system is not divided into unconnected parts. Fault-tolerance by stabilization ensures the continued availability of systems by correctly restoring the system state whenever the system exhibits incorrect behavior due to the occurrence of faults. Obviously, such a self-organizing behavior requires some assumptions about the occurrence of faults.

When talking about stabilization there is always an environment that acts as an adversary producing transient faults thus affecting the consistency of variables upon which the actual stabilization depends. In most publications on stabilization, see e.g. [2, 7, 13], the adversary is not explicitly modeled. The influence of faults is indirectly taken into account by considering *arbitrary initial states* in stabilizing systems. This models the situation that a fault-action just occurred and now time is given to the stabilizing system to reestablish a legal state.

From a more application oriented point of view stabilizing systems are constructed to be non-initializing because [24]:

- (re-)initialization of distributed systems is a very difficult task, cf. [6], unless one constructs systems which can start in arbitrary states, and
- the main structuring principle to manage the complexity involved in the design and verification of these systems is to build complex stabilizing systems as a particular composition of simpler stabilizing systems, see e.g. [6, 13, 15, 16]. In order for this composition to yield a stabilizing overall system, the simpler systems have to be non-initializing (see [25] for details).

So, from now on we restrict our attention to the set $\mathcal{NI}$ of non-initializing systems, formally defined as $\mathcal{NI} \stackrel{\text{def}}{=} \{S \in \mathcal{S} \mid\, \models \Theta_S \leftrightarrow \mathit{true}\}$.

3.2 Formal Definition of Stabilization

Arora and Gouda advocate in [4, 5] a general and uniform definition of fault-tolerance based on the terms *convergence* and *closure*. These two properties characterize behavior of all stabilizing systems.

The observation which led to such a uniform definition is that there are two distinct kinds of behavior that a fault-tolerant system displays. As long as no fault occurs it performs the task that it is designed for, such as providing mutual exclusion, updating routing tables, or load balancing. As soon as a fault occurs a concerted effort of the processes causes the whole system to re-establish a so-called *legal* state [12] from which point on it displays the original fault free behavior. So, in [4, 5] it is assumed that for a fault-tolerant system there exists a predicate *le* characterizing the set of legal system states which is invariant throughout fault-free system execution.

Furthermore Arora and Gouda state that faults can uniquely be represented as actions that upon execution perturb the system state [11]. Instead of anticipating various possible faults, it is only assumed that there exists a predicate *fs* weaker than *le*, called the *fault span* of the system [4], which defines the extent to which fault actions may perturb the legal states during system execution. Then they define that a system $S \in \mathcal{NI}$ is *stabilizing* w.r.t. fs and le iff it has the following two properties:

– *convergence*: Starting in an arbitrary state where fs holds, eventually a state is reached where le holds.
– *closure*: Predicates fs and le are invariant under execution of actions from S.

Based on this definition of stabilization Arora and Gouda define that a system $S \in \mathcal{NI}$ is *fault-tolerant* w.r.t. an adversary A and predicate le iff there exists a fault-span fs such that:

1. System S is stabilizing w.r.t. fs and le, and
2. predicate fs is invariant under execution of both actions from S and A.

Typical problems tackled in fault-tolerant computing by stabilization include communication protocols, clock synchronization, mutual exclusion, distributed reset, load balancing, spanning tree constructions and leader election (see [25] for an extensive list of references).

The above informal description of stabilization can be formally captured by means of LTL$^-$ as follows:

Definition 4. Given fts $S \in \mathcal{NI}$, fault span fs and predicate le, with $\models le \rightarrow fs$. System S is *stabilizing* w.r.t. fs and le iff S satisfies the following properties:

convergence: $S \models \Box(fs \rightarrow \Diamond le)$
closure: $S \models \Box(fs \rightarrow \Box fs)$
$S \models \Box(le \rightarrow \Box le)$

□

A formal definition of fault-tolerance w.r.t. an adversary and a set of legal states is straight-forward but requires a definition of parallel composition on fts's which we omitted due to space limitations (see [25]). However it is worth noting that the forthcoming proof rules also apply in this case.

Properties of the form $\Box(p \rightarrow \Diamond q)$ are usually termed *response properties* in temporal logic [20]. We deviate from this convention and call them *convergence properties* since this is the term used in the stabilization community.

4 Proof Rules for Convergence and Closure

Despite the intricacy of stabilizing systems the mathematical foundations for their design and verification have hardly been investigated. In this section we present a collection of temporal proof rules which exploit the fact that stabilizing systems are non-initializing and that their fault-span is not left by system transitions, i.e. the fault-span is closed under system execution.

We give proof rules for program validity of formulas $\Box(p \rightarrow \Box p)$ and $\Box(p \rightarrow \Diamond q)$ for state predicates p, q. Repeated application of these rules gives a complete reduction of the program validity of these formulas into a set of assertional validities [20]. We prove relative completeness of our rules, i.e. we assume that there exists an oracle that provides proofs or otherwise verifies all generally valid assertions [19]; stated differently, we assume all valid assertions as axioms.

There exist complete proof rules for closure and convergence for arbitrary fts's [21]. As explained, rather than simply recalling these very powerful rules, we adapt the general rules to deal with systems from set $\mathcal{NI}$.

In the following presentation we always state the general proof rule first, followed by the adapted rule for set $\mathcal{NI}$.

Note, that we do not have to deal with any past-formulas. There is nothing to prove about the past of a stabilizing systems since any of its states may have no past.

4.1 Proof Rules for Closure Properties

We start with a complete rule for proving $S \models \Box(p \rightarrow \Box p)$ for $S \in \mathcal{S}$ and some state predicate p. The rule for general invariance in [19, 21] refers to past modalities; in order to avoid the introduction of past-modalities we state a rule which only refers to future modalities.

$$\begin{array}{l} p \Rightarrow \varphi \\ \varphi \rightarrow p \\ \{\varphi\}\ T\ \{\varphi\} \\ \hline \Box(p \rightarrow \Box p) \end{array} \qquad \mathsf{GClos}$$

Concerning notation: In the following rules we assume that a fixed fts is given and refer to its components where necessary (e.g. to its set of transitions T in the rule above), cf. [19]. Recall, that operator $\Rightarrow$ denotes entailment $\varphi \Rightarrow \psi \stackrel{\text{def}}{=} \Box(\varphi \rightarrow \psi)$, whereas $\rightarrow$ denotes usual implication between assertions or formulas. Soundness and completeness of the above rule means that assertion p is closed in system S iff there exists a state formula φ such that

1. p implies φ in every reachable state of S,
2. φ implies p in every state, and
3. φ is invariant under transitions of S.

The first two premises imply that φ and p are equivalent on the set of reachable states of S. Typically, the additional state formula φ is used to characterize the set of p-states, i.e. states where predicate p holds, that are reachable in the system under investigation. Finding the appropriate assertion φ, i.e. coding enough reachability information into such an auxiliary predicate, is the intricate part in the application of rule GClos.

Proposition 5. *Rule* GClos *is sound and complete for proving* $S \models \Box(p \rightarrow \Box p)$ *for* $S \in \mathcal{S}$. □

Remark. Replacing in the second premise of rule GClos p by q actually yields a complete proof rule for the more general case $S \models \Box(p \rightarrow \Box q)$. Completeness of this rule is proven along the lines of the completeness proof for rule GClos which is sketched below.

Proof. As explained, we consider completeness of our proof rules relative to assertional validity. Hence it is sufficient to show that validity of the premises follows from validity of the conclusion. So, given a system S and predicate p such that $S \models \Box(p \rightarrow \Box p)$ we have to define predicate φ such that the premises of rule GClos becomes valid.

Assuming that the data domain of the assertion language is expressive enough to encode records of data and lists of records one can define a state assertion χ that holds in a state

s iff s is reachable in S, i.e. appears in some computation of S (for technical details of this definition see [19]).

Now, we consider as auxiliary predicate $\chi_p \stackrel{\mathrm{def}}{=} \chi \wedge p$ and prove validity of the premises for predicate χ_p.

$S \models p \Rightarrow \chi_p$**:** Given a computation $\sigma \in [\![S]\!]$ and position i such that $\sigma_i \models p$. Obviously σ_i is reachable and satisfies p so $\sigma_i \models \chi_p$.

$\models \chi_p \rightarrow p$**:** By the definition of χ_p we have $\models \chi_p \rightarrow p$ because p is a conjunct of χ_p.

$\models \{\chi_p\}\ T\ \{\chi_p\}$**:** We prove that whenever state s satisfies predicate χ_p every t-successor s' of s, for $t \in T$, also satisfies χ_p.

	$s \models \chi_p$	
iff	$s \models \chi \wedge p$	by definition of χ_p
iff	$\exists \iota \in Pref(S).(last(\iota) = s \wedge s \models p)$	by definition of χ

Let ι be a computation prefix leading to s, i.e. $last(\iota) = s$. We have that $\iota \circ \langle s' \rangle \in Pref(S)$ since s' is by assumption a t-successor of s for some $t \in T$ and since S is machine closed. Since $s \models p$ and $S \models \Box(p \rightarrow \Box p)$ we have $s' \models p$. We conclude $s' \models \chi \wedge p$, i.e. $s' \models \chi_p$.

□

The adapted complete rule for proving closure for systems $S \in \mathcal{NI}$ does not use any auxiliary predicate.

$$\frac{\{p\}\ T\ \{p\}}{\Box(p \rightarrow \Box p)} \qquad \textsf{Clos-}\mathcal{NI}$$

The remaining premise $\{p\}\ T\ \{p\}$ is local in the sense that it does not require any more temporal reasoning; it is purely assertional. Note, that this rule is not complete if we consider arbitrary fts's.

Instead of proving directly soundness and completeness of rule Clos-$\mathcal{NI}$ for systems from $\mathcal{NI}$, we prove that $S \models \Box(p \rightarrow \Box p)$ (for $S \in \mathcal{NI}$) is derivable by rule GClos iff it is derivable by rule Clos-$\mathcal{NI}$. This reveals how we obtained the premise of rule Clos-$\mathcal{NI}$, namely by an explicit characterization of predicate φ in rule GClos and subsequent simplification of its premises.

Proposition 6. *Given system $S \in \mathcal{NI}$. $S \models \Box(p \rightarrow \Box p)$ is derivable by rule* GClos *iff it is derivable by rule* Clos-$\mathcal{NI}$*.*

□

Proof. Given system $S \in \mathcal{NI}$ and predicate p. We just prove the more interesting direction: validity of the premises of rule GClos implies validity of the premise of rule Clos-$\mathcal{NI}$. So assume that there exists a predicate φ such that $S \models p \Rightarrow \varphi, \models \varphi \rightarrow p$ and $\models \{\varphi\}\ T\ \{\varphi\}$ holds. We have to prove that $\models \{p\}\ T\ \{p\}$ holds. We exploit the soundness and completeness result of rule GClos. From the assumptions we have by soundness of rule GClos that $S \models \Box(p \rightarrow \Box p)$. From the completeness proof for rule GClos we know that $\models \{\chi_p\}\ T\ \{\chi_p\}$ holds. Now it suffices to note that $\models \chi_p \leftrightarrow p$ for systems $S \in \mathcal{NI}$ since:

	$s \models \chi_p$	
iff	$s \models \chi \wedge p$	by definition of χ_p
iff	$\exists \iota \in Pref(S).last(\iota) = s \wedge s \models p$	by definition of χ_p
iff	$s \models p$	since $S \in \mathcal{NI}$

The last step is the reason why we can do without the auxiliary predicate φ in rule Clos-$\mathcal{NI}$: every p-state of S is reachable anyhow since $S \in \mathcal{NI}$ so predicate χ_p and p are equivalent.

From Proposition 5 and 6 we get:

Corollary 7. *Rule* Clos-$\mathcal{NI}$ *is sound and complete for proving* $S \models \Box(p \rightarrow \Box p)$ *for* $S \in \mathcal{NI}$. □

Before we present proof rules for convergence we state a proposition which is important for the simplification of forthcoming proof rules.

Proposition 8. *For all systems* $S \in \mathcal{NI}$ *and all temporal formulas* $\varphi \in LTL^-$ *the following holds:*

$S \models \varphi$ *iff* $S \models \Box\varphi$

□

The proof of this proposition is based on the observation that the set of computations of all systems $S \in \mathcal{NI}$ is suffix closed.

4.2 Proof Rules for Convergence

In the presentation of the proof rules for convergence we also consider subsets of $\mathcal{NI}$, consisting of all those systems which preserve a predefined state predicate p, i.e. sets of the form $\mathcal{NI}_p \stackrel{\text{def}}{=} \{S \in \mathcal{NI} \mid S \models \Box(p \rightarrow \Box p)\}$. The relevance of $\mathcal{NI}_p$ stems from the fact that we are interested in establishing the stabilization of a given system $S \in \mathcal{NI}$ w.r.t. some state predicates p, q. So we have to prove $S \models \Box(p \rightarrow \Diamond q) \wedge \Box(p \rightarrow \Box p) \wedge \Box(q \rightarrow \Box q)$. Proving $S \models \Box(p \rightarrow \Box p)$ first (by rule Clos-$\mathcal{NI}$), we have shown that S belongs to set $\mathcal{NI}_p$ and thus we can apply the rules for $\mathcal{NI}_p$ in order to establish the convergence of S.

The existing rules for proving convergence properties are usually partitioned into single-step convergence rules and extended convergence rules. We keep this pattern and mainly follow the presentation in [19]. Similar to the previous section we first state the general rule, followed by an adapted rule for set $\mathcal{NI}$ and finally the rule for $\mathcal{NI}_p$.

Single-step Convergence Rule under Weak Fairness Single-step convergence rules are applicable in case that there exists at least one transition that accomplishes the desired convergence within one step. We obtain two slightly different rules, depending on whether this so-called helpful transition is executed weakly or strongly fair. The general proof rule as stated in [19] is:

$$\frac{\begin{array}{l} p \Rightarrow (q \vee \varphi) \\ \{\varphi\}\ T\ \{q \vee \varphi\} \\ \{\varphi\}\ t\ \{q\} \\ \varphi \Rightarrow (q \vee en(t)) \end{array}}{\Box(p \rightarrow \Diamond q)} \qquad \mathsf{WConv}$$

In this rule t identifies the helpful transition, contained in the set of weakly fair executed transitions, and $en(t)$ denotes the enabledness of t. The rule states that we have to find a state predicate φ such that p entails $q \vee \varphi$. Predicate φ has to be preserved by every transition in T unless q is established. Since the helpful transition t is enabled in φ-states if q is not yet established, we conclude that either q is established by a T-transition or, by weak fairness, finally t is executed which also establishes q.

When restricting our attention to systems in the set $\mathcal{NI}$ we get due to Proposition 8 the following simpler proof rule.

$$\frac{\begin{array}{l} p \to (q \vee \varphi) \\ \{\varphi\}\ T\ \{q \vee \varphi\} \\ \{\varphi\}\ t\ \{q\} \\ \varphi \to (q \vee en(t)) \end{array}}{\Box(p \to \Diamond q)} \qquad \mathsf{WConv}\text{-}\mathcal{NI}$$

What looks like a minor change, replacing two times entailment by implication, constitutes in fact a considerable saving. Whereas the entailment properties of rule WConv require in general further invariants for their proof, in case of stabilizing systems we are done with proving ordinary implications!

Proposition 9. *Given system $S \in \mathcal{NI}$. $S \models \Box(p \to \Diamond q)$ is derivable by rule* WConv *iff it is derivable by rule* WConv-$\mathcal{NI}$.

□

In case that the system under consideration belongs to set $\mathcal{NI}_p$ we get a further simplification.

$$\frac{\begin{array}{l} \{p \wedge \neg q\}\ t\ \{q\} \\ p \to (q \vee en(t)) \end{array}}{\Box(p \to \Diamond q)} \qquad \mathsf{WConv}\text{-}\mathcal{NI}_p$$

This simplification causes no loss of generality when considering systems in $\mathcal{NI}_p$.

Proposition 10. *Given system $S \in \mathcal{NI}_p$. $S \models \Box(p \to \Diamond q)$ is derivable by rule* WConv *iff it is derivable by rule* WConv-$\mathcal{NI}_p$.

□

Proof. The proof of this proposition is based on the following two lemmata which are proved in [25]. These lemmata refer to a predicate χ_p^q which is a variant of predicate χ_p used in the completeness proof of Proposition 5. Predicate χ_p^q is defined such that $s \models \chi_p^q$ holds in a state $s \in \Sigma_S$ iff there exists a reachable p-state s' and a computation segment ι leading from s' to s, s.t. ι does not contain any q-states.

Lemma 11. *Given system $S \in \mathcal{S}$ and predicates p, q, φ. If $S \models p \Rightarrow (q \vee \varphi)$ and $\models \{\varphi\}\ T\ \{q \vee \varphi\}$ holds then $\models \chi_p^q \to \varphi$ and $\{\chi_p^q\}\ T\ \{q \vee \chi_p^q\}$.*

□

The second lemma gives a characterization of predicate χ_p^q for systems $S \in \mathcal{NI}_p$.

Lemma 12. *For systems $S \in \mathcal{NI}_p$ and predicates p, q we have $\models \chi_p^q \leftrightarrow (p \wedge \neg q)$.*

□

Using these lemmata it is not difficult to complete the proof for both directions of Proposition 10.

Single-step Convergence Rule under Strong Fairness The second single-step convergence rule relies on the existence of a helpful transition t in the set of strongly fair executed transitions. The general rule is:

$$\frac{\begin{array}{l} p \Rightarrow (q \vee \varphi) \\ \{\varphi\}\ T\ \{q \vee \varphi\} \\ \{\varphi\}\ t\ \{q\} \\ \varphi \Rightarrow \Diamond(q \vee en(t)) \end{array}}{\Box(p \rightarrow \Diamond q)} \qquad \mathsf{SConv}$$

As in rule WConv, t denotes the helpful transition, but now contained in the set of strongly fair executed transitions. Only the fourth premise is changed. We have to prove that φ entails that eventually $q \vee en(t)$ holds.

With the same justification as in the case of weak fairness we obtain the following two simplifications of this rule:

$$\frac{\begin{array}{l} p \rightarrow (q \vee \varphi) \\ \{\varphi\}\ T\ \{q \vee \varphi\} \\ \{\varphi\}\ t\ \{q\} \\ \varphi \rightarrow \Diamond(q \vee en(t)) \end{array}}{\Box(p \rightarrow \Diamond q)} \qquad \mathsf{SConv}\text{-}\mathcal{NI}$$

We have the corresponding relative completeness result for rule SConv-$\mathcal{NI}$ w.r.t. rule SConv.

Proposition 13. *Given system $S \in \mathcal{NI}$. $S \models \Box(p \rightarrow \Diamond q)$ is derivable by rule* SConv *iff it is derivable by rule* SConv-$\mathcal{NI}$.

□

In case that the system under consideration belongs to set $\mathcal{NI}_p$ we get:

$$\frac{\begin{array}{l} \{p \wedge \neg q\}\ t\ \{q\} \\ p \rightarrow \Diamond(q \vee en(t)) \end{array}}{\Box(p \rightarrow \Diamond q)} \qquad \mathsf{SConv}\text{-}\mathcal{NI}_p$$

The expected result for $\mathcal{NI}_p$ in case of strong fairness is:

Proposition 14. *Given system $S \in \mathcal{NI}_p$. $S \models \Box(p \rightarrow \Diamond q)$ is derivable by rule* SConv *iff it is derivable by rule* SConv-$\mathcal{NI}_p$.

□

As can be observed in most convergence proofs for stabilizing systems, *concerted effort* of several transitions is generally necessary to establish q after a p-state has been encountered. So more powerful rules for so called *extended* convergence are needed.

Extended Convergence Rules The general proof rule reduces extended convergence properties to a set of single-step convergence properties. These single-step convergence properties commonly serve to establish a well founded induction argument. We follow the presentation in [19].

A binary relation $\preceq$ over set A is a pre-order if it is reflexive and transitive. If $a \preceq b$ but not $b \preceq a$ we say that a precedes b denoted by $a \prec b$. The irreflexive, asymmetric, and transitive ordering $(A, \prec)$ induced by $(A, \preceq)$ is *well-founded* if there does not exist an infinite sequence $\langle a_0, a_1, a_2, \ldots\rangle$ where $a_i \in A$ and $a_{i+1} \prec a_i$ for all $i \geq 0$. A pre-order $\preceq$ is called well-founded if its induced ordering $\prec$ is well-founded. Since we do not deal with past-operators we can use ranking functions which map states, rather than computation prefixes, to a well founded domain. So in the following rule from [19] $\delta : \Sigma \mapsto A$ is a ranking function where Σ is the state space of the system under consideration, and A the domain of a well-founded pre-order $(A, \preceq)$. Then, the following rule can be used to prove convergence:

$$\frac{\begin{array}{l} p \Rightarrow (q \vee \varphi) \\ (\varphi \wedge \delta = a) \Rightarrow \Diamond(q \vee (\varphi \wedge \delta \prec a)) \end{array}}{\Box(p \rightarrow \Diamond q)} \qquad \mathsf{EConv\text{-}1}$$

The first premise ensures that $q \vee \varphi$ holds at a position in a computation if p holds at that position. The second premise ensures that each reachable state where φ holds is eventually followed by a φ-state with a lower rank or a state where q holds.

Proposition 15 (Manna & Pnueli). *Rule set* { EConv-1, WConv, SConv} *is sound and complete for proving* $S \models \Box(p \rightarrow \Diamond q)$ *for* $S \in \mathcal{S}$ *and predicates* p, q.

$\Box$

Simplifications of rule EConv-1 are obtained along the same lines as in the case of the single-step convergence rules.

$$\frac{\begin{array}{l} p \rightarrow (q \vee \varphi) \\ (\varphi \wedge \delta = a) \rightarrow \Diamond(q \vee (\varphi \wedge \delta \prec a) \end{array}}{\Box(p \rightarrow \Diamond q)} \qquad \mathsf{EConv\text{-}1\text{-}}\mathcal{NI}$$

Proposition 16. *Given system* $S \in \mathcal{NI}$. $S \models \Box(p \rightarrow \Diamond q)$ *is derivable by rule* EConv-1 *iff it is derivable by rule* EConv-1-$\mathcal{NI}$.

$\Box$

As corollary of Proposition 15 and 16 we get:

Corollary 17. *Rule set* { EConv-1-$\mathcal{NI}$, WConv-$\mathcal{NI}$, SConv-$\mathcal{NI}$} *is sound and complete for proving* $S \models \Box(p \rightarrow \Diamond q)$ *for* $S \in \mathcal{NI}$ *and predicates* p, q.

$\Box$

In case of set $\mathcal{NI}_p$ we get the following adapted rule:

$$\frac{(p \wedge \neg q \wedge \delta = a) \rightarrow \Diamond(q \vee \delta \prec a)}{\Box(p \rightarrow \Diamond q)} \qquad \mathsf{EConv\text{-}1\text{-}}\mathcal{NI}_p$$

We have the following proposition.

Proposition 18. *Given system* $S \in \mathcal{NI}_p$. $S \models \Box(p \rightarrow \Diamond q)$ *is derivable by rule* EConv-1 *iff it is derivable by rule* EConv-1-$\mathcal{NI}_p$.

$\Box$

From Proposition 15 and 18 we get the following soundness and completeness result for proving convergence properties of systems from $\mathcal{NI}_p$.

Corollary 19. *Rule set* { EConv-1-$\mathcal{NI}_p$, WConv-$\mathcal{NI}_p$, SConv-$\mathcal{NI}_p$} *is sound and complete for proving* $S \models \Box(p \rightarrow \Diamond q)$ *for* $S \in \mathcal{NI}_p$ *and predicates* p, q.

$\Box$

Extended Convergence with Helpful Directions In [19] a strategy is proposed to replace the premise concerning convergence in rule EConv-1 by applying a combination of single-step convergence rules. The resulting rule, stated below, uses the set $WF \cup SF$ containing all weakly or strongly fair executed transitions. Without loss of generality $WF \cap SF = \emptyset$ holds. Furthermore the rule assumes the existence of a set of state formulas $\{\varphi_1, \ldots, \varphi_n\}$, each φ_i corresponding to one $t_i \in WF \cup SF$, a well-founded pre-order $(A, \preceq)$, and a ranking function with range A. Let $\varphi \stackrel{\text{def}}{=} \bigvee_{i=1}^{n} \varphi_i$.

$$\frac{\begin{array}{l} p \Rightarrow (q \vee \varphi) \\ \{\varphi_i \wedge \delta = a\}\ T\ \{q \vee (\varphi \wedge \delta \prec a) \vee (\varphi_i \wedge \delta \preceq a)\} \\ \{\varphi_i \wedge \delta = a\}\ t_i\ \{q \vee (\varphi \wedge \delta \prec a)\} \\ \varphi_i \Rightarrow (q \vee en(t_i)) \quad \text{for } t_i \in WF \\ \varphi_i \Rightarrow \Diamond(q \vee en(t_i)) \quad \text{for } t_i \in SF \end{array}}{\Box(p \rightarrow \Diamond q)} \qquad \textsf{EConv-2}$$

In φ_i-states transition t_i is the helpful transition contained in $WF \cup SF$. The third premise guarantees that t_i, executed in a state where φ_i holds, either decreases the rank while preserving φ or establishes q. The other transitions in T either have to preserve these φ_i or have to lower the rank or establish q. The fourth and fifth premise guarantee that the t_i transitions are eventually executed when a φ_i-state has been encountered.

Proposition 20 (Manna & Pnueli). *Rule* EConv-2 *is sound and complete for proving* $S \models \Box(p \rightarrow \Diamond q)$ *for* $S \in \mathcal{S}$ *and state predicates* p, q.

□

In case we restrict our attention to systems in set $\mathcal{NI}$ we obtain the following simplified rule:

$$\frac{\begin{array}{l} p \rightarrow (q \vee \varphi) \\ \{\varphi_i \wedge \delta = a\}\ T\ \{q \vee (\varphi \wedge \delta \prec a) \vee (\varphi_i \wedge \delta \preceq a)\} \\ \{\varphi_i \wedge \delta = a\}\ t_i\ \{q \vee (\varphi \wedge \delta \prec a)\} \\ \varphi_i \rightarrow (q \vee en(t_i)) \quad \text{for } t_i \in WF \\ \varphi_i \rightarrow \Diamond(q \vee en(t_i)) \quad \text{for } t_i \in SF \end{array}}{\Box(p \rightarrow \Diamond q)} \qquad \textsf{EConv-2-}\mathcal{NI}$$

The corresponding proposition states:

Proposition 21. *Given system* $S \in \mathcal{NI}$. $S \models \Box(p \rightarrow \Diamond q)$ *is derivable by rule* EConv-2 *iff it is derivable by rule* EConv-2-$\mathcal{NI}$.

□

From Proposition 20 and 21 we obtain:

Corollary 22. *Rule* EConv-2-$\mathcal{NI}$ *is sound and complete for proving* $S \models \Box(p \rightarrow \Diamond q)$ *for* $S \in \mathcal{NI}$ *and state predicates* p, q.

□

Finally, we get the following rule for extended convergence in case of $\mathcal{NI}_p$ where the φ_i are selected such that $\models (p \wedge \neg q) \rightarrow \bigvee_{i=1}^{n} \varphi_i$ holds.

$$\frac{\begin{array}{ll}\{\varphi_i \wedge \delta = a\}\ T\ \{q \vee \delta \prec a \vee (\varphi_i \wedge \delta \preceq a)\} & \\ \{\varphi_i \wedge \delta = a\}\ t_i\ \{q \vee \delta \prec a\} & \\ \varphi_i \rightarrow (q \vee en(t_i)) \quad \text{for } t_i \in WF & \\ \varphi_i \rightarrow \Diamond(q \vee en(t_i)) \quad \text{for } t_i \in SF & \end{array}}{\Box(p \rightarrow \Diamond q)} \qquad \mathsf{EConv\text{-}2\text{-}}\mathcal{NI}_p$$

Proposition 23. *Given system $S \in \mathcal{NI}_p$. $S \models \Box(p \rightarrow \Diamond q)$ is derivable by rule* EConv-2 *iff it is derivable by rule* EConv-2-$\mathcal{NI}_p$*.*

□

As in the previous case we get from Proposition 20 and 23:

Corollary 24. *Rule* EConv-2-$\mathcal{NI}_p$ *is sound and complete for proving $S \models \Box(p \rightarrow \Diamond q)$ for $S \in \mathcal{NI}_p$ and state predicates p, q.*

□

We have obtained rule EConv-2-$\mathcal{NI}_p$ by replacing φ in rule EConv-2 by $p \wedge \neg q$ and perform subsequent simplifications exploiting the closure of p. However, it is not possible to eliminate the auxiliary predicates φ_i in rule EConv-2-$\mathcal{NI}_p$; the construction of the φ_i's in the completeness proof of rule EConv-2 [19] reveals that there does not exist a general characterization of the φ_i by means of the predicates p and q.

5 Conclusion

In this paper, we presented a formal framework for the analysis of stabilizing systems, which play an important role in the field of fault-tolerance. Temporal logic has been used to define the central notions underlying stabilization such as closure and convergence. Indeed, temporal logic turns out to be a natural formalism to define these notions. Then, we used the temporal approach to program verification to derive a set of proof rules which can be used to analyze stabilizing systems. Using the semantical features of these systems, such as the fact that they are non-initializing, we obtain a set of proof rules which are simpler than the general temporal proof rules. These proof rules provide a basis for computer aided verification of stabilizing systems.

In [25] we have completed the list of proof rules for stabilizing systems by giving rules for *pseudo-stabilization* [10] and also some useful temporal tautologies which can be used in the verification of actual systems. Furthermore we present in [25] a collection of design rules for the formal development of stabilizing systems based on the above presented verification rules.

Acknowledgments: We thank Amir Pnueli and the anonymous referees for valuable comments.

References

1. M. Abadi and L. Lamport. The existence of refinement mappings. *Theoretical Computer Science*, 82(2), 1991.
2. Y. Afek and G.M. Brown. Self-stabilization over unreliable communication media. *Distributed Computing*, 7:27–34, 1993.
3. R. Alur, T. Henzinger, and P. Ho. Automatic symbolic model checking of embedded systems. In *IEEE Real-Time Systems Symposium*, 1993.

4. A. Arora. *A Foundation of Fault Tolerant Computing.* PhD thesis, The University of Texas at Austin, 1992.
5. A. Arora and M.G. Gouda. Closure and convergence: a foundation of fault-tolerant computing. *IEEE Transactions on Software Engineering*, 19:1015–1027, 1993.
6. A. Arora and M.G. Gouda. Distributed reset. *IEEE Transactions on Computers*, 43:1026–1038, 1994.
7. J. Beauquier and S. Delaët. Probabilistic self-stabilizing mutual exclusion in uniform rings. In *PODC94 Proceedings of the Thirteenth Annual ACM Symposium on Principles of Distributed Computing*, page 378, 1994.
8. R. S. Boyer and J. S. Moore. Integrating decision procedures into heuristic theorem provers. *Machine Intelligence*, 11, 1986.
9. J. Burch, E. Clarke, K. McMillan, D. Dill, and L. Hwang. Symbolic model checking: 10^{20} states and beyond. In *Logic in Computer Science*, 1990.
10. J.E. Burns, M.G. Gouda, and R.E. Miller. Stabilization and pseudo-stabilization. *Distributed Computing*, 7:35–42, 1993.
11. F. Cristian. A rigorous approach to fault-tolerant programming. *IEEE Transactions on Software Engineering*, 11(1), 1985.
12. E.W. Dijkstra. Self stabilizing systems in spite of distributed control. *Communications of the ACM*, 17(11), 1974.
13. S. Dolev, A. Israeli, and S. Moran. Self-stabilization of dynamic systems assuming only read/write atomicity. *Distributed Computing*, 7:3–16, 1993.
14. M.G. Gouda, R.R. Howell, and L.E. Rosier. The instability of self-stabilization. *Acta Informatica*, 27:697–724, 1990.
15. M.G. Gouda and N. Multari. Stabilizing communication protocols. *IEEE Transactions on Computers*, 40:448–458, 1991.
16. S. Katz and K.J. Perry. Self-stabilizing extensions for message-passing systems. *Distributed Computing*, 7:17–26, 1993.
17. L. Lamport. Solved problems, unsolved problems, and non-problems in concurrency. In *Proceedings of the 3rd Annual ACM Symposium on Principles of Distributed Computing*, 1984.
18. C. Lin and J. Simon. Possibility and impossibility results for self-stabilizing phase clocks on synchronous rings. In *Proceedings of the Second Workshop on Self-Stabilizing Systems*, pages 10.1–10.15, 1995.
19. Z. Manna and A. Pnueli. Completing the temporal picture. *Theoretical Computer Science*, 83(1), 1991.
20. Z. Manna and A. Pnueli. *The Temporal Logic of Reactive and Concurrent Systems.* Springer Verlag, 1991.
21. Z. Manna and A. Pnueli. *Temporal Verification of Reactive Systems.* Springer Verlag, 1995.
22. S. Owre, J. Rushby, N. Shankar, and F. von Henke. Formal verification for fault-tolerant architectures: Some lessons learned. In *FME '93:Industrial-Strength Formal Methods*, number 670 in LNCS. Springer Verlag, 1993.
23. S. Owre, J.M. Rushby, and N. Shankar. Pvs: a prototype verification system. In *11th Int. Conf. on Automated Deduction (CADE)*, volume 607 of *LNCS*. Springer Verlag, 1992.
24. M. Schneider. Self-stabilization. *ACM Computing Surveys*, 25:45–67, 1993.
25. M. Siegel. *Phased Design and Verifcation of Stabilizing Systems.* PhD thesis, University of Kiel, 1996.