Verification by Augmented Finitary Abstraction*

Yonit Kesten† Amir Pnueli‡

February 7, 2001

Abstract. The paper deals with the proof method of *verification by finitary abstraction* (VFA), which presents a feasible approach to the verification of the temporal properties of (potentially infinite-state) reactive systems. The method consists of a two-step process by which, in a first step, the system and its temporal specification are jointly abstracted into a finite-state system and a finite-state specification. The second step uses model checking to establish the validity of the abstracted property over the abstracted system.

The VFA method can be considered as a viable alternative to verification by temporal deduction which, up to now, has been the main method generally applicable for verification of infinite-state systems.

The paper presents a general recipe for the joint abstraction, which is shown to be *sound*, where soundness means that validity over the abstract system implies validity over the concrete (original) system. To make the method applicable for the verification of liveness properties, pure abstraction is sometimes no longer adequate. We show that by augmenting the system by an appropriate (and standardly constructible) *progress monitor*, we obtain an augmented system, whose computations are essentially the same as the original system, and which may now be abstracted while preserving the desired liveness properties. We refer to the extended method as *verification by augmented abstraction* (VAA).

We then proceed to show that the VAA method is sound and complete for proving all properties expressible by temporal logic (including both safety and liveness). Completeness establishes that whenever the property is valid, there exists a finitary abstraction which abstracts the system, augmented by an appropriate progress monitor, into a finite-state system which validates the abstracted property.

1 Introduction

When verifying temporal properties of reactive systems, the common wisdom is: if it is finite-state, model-check it, otherwise one must use temporal deduction, supported by

*This research was supported in part by the Minerva Center for Verification of Reactive Systems, a gift from Intel, a grant from the U.S.-Israel bi-national science foundation, and an *Infrastructure* grant from the Israeli Ministry of Science and the Arts.

†Contact author: Dept. of Communication Systems Engineering, Ben Gurion University, Beer-Sheva, Israel, ykesten@bgumail.bgu.ac.il

‡Dept. of Applied Mathematics and Computer Science, the Weizmann Institute of Science, Rehovot, Israel, amir@wisdom.weizmann.ac.il

theorem provers such as STeP, PVS, etc.

The study of abstraction as an aid to verification demonstrated that, in some interesting cases, one can abstract an infinite-state system into a finite-state one. This suggests an alternative approach to the temporal verification of infinite-state systems: abstract first and model check later.

In this work, we present a general framework based on linear temporal logic for a joint abstraction of a reactive system $\mathcal{D}$ and its specification expressed as a linear temporal logic (LTL) formula ψ. The unique features of this abstraction method is that it takes full account of all the fairness assumptions (including strong fairness) associated with the system $\mathcal{D}$ and can, therefore, establish liveness properties, in contrast to most other abstraction approaches that can only support verification of safety properties.

We first provide a sound recipe for the application of the method of *verification by finitary abstraction* (VFA). That is, given an arbitrary state mapping α which maps concrete to abstract states, we show how to define the abstract versions S^α and ψ^α such that $S^\alpha \models \psi^\alpha$ implies $S \models \psi$, establishing that ψ is S-valid. In the case that α maps all concrete variables into abstract variables ranging over finite domains, S^α will be a finite-state system, and $S^\alpha \models \psi^\alpha$ can be verified by model checking. An earlier version of this part of the presentation appeared in [KP98b].

Applying the method of finitary abstraction for the proofs of liveness properties, we find that, sometimes, pure abstraction is no longer adequate. For these cases, it is possible to construct an additional module M, to which we refer as a *progress monitor*, such that the augmented system $\mathcal{D} \,|||\, M$ (the synchronous parallel composition of $\mathcal{D}$ and M) has essentially the same set of computations as the original $\mathcal{D}$ and can be abstracted in a way which preserves the desired liveness property. We refer to this extended proof method as the method of *verification by augmented abstraction* (VAA).

In Section 7 we show that the VAA method is sound. That is, for every abstraction mapping α, if the abstracted property ψ^α is valid over the abstracted augmented system $\mathcal{D} \,|||\, M$, and the monitor M does not constrain the computations of $\mathcal{D}$ (effective sufficient conditions for this are provided), then we can safely infer $\mathcal{D} \models \psi$.

Sections 8—9 are dedicated to the proof of *completeness* of the VAA method. We show that if ψ is valid over $\mathcal{D}$, then there exist a monitor M which does not constrain the computations of $\mathcal{D}$ and a finitary abstraction mapping α, such that $(\mathcal{D} \,|||\, M)^\alpha \models \psi^a$.

As will be shown in the next subsection, the idea of using abstraction for simplifying the task of verification is certainly not new with us. Even the observation that, in many interesting cases, infinite-state systems can be abstracted into finite-state systems which can be model checked has been made before. The main contributions of the paper can be summarized as

- Reformulation of the main principles underlying abstraction for the simpler cases of a linear (LTL) framework and a functional abstraction mapping (instead of the more general abstraction relation, leading to the full Galois connection theory).

- Consideration of the powerful computational model of *fair discrete systems* (FDS) which incorporates full fairness (including weak and strong fairness) and showing how to perform a joint abstraction of a system and its specification, which can be an arbitrary LTL formula.

- Observing that for some verification tasks involving liveness, pure abstraction is inadequate, and devising the method of *verification by augmented abstraction*.

- Establishing completeness of the VAA method.

1.1 Related Work

There has been an extensive study of the use of data abstraction techniques, mostly based on the notions of *abstract interpretation* ([CC77], [CH78]). Most of the previous work was done in a branching context which complicates the problem if one wishes to preserve both existential and universal properties. On the other hand, if we restrict ourselves to a universal fragment of the logic, e.g. ACTL*, then the conclusions reached are similar to our main result for the restricted case that the property ψ contains negations only within assertions.

The paper [CGL94] obtains a similar result for the fragment ACTL*. However, instead of starting with a concrete property ψ and abstracting it into an appropriate ψ^α, they start with an abstract ACTL* formula Ψ evaluated over the abstract system $\mathcal{D}^\alpha$ and show how to translate (concretize) it into a concrete formula $\psi = \mathcal{C}(\Psi)$. The concretization is such that $\alpha^-(\psi) = \Psi$.

The survey in [CGL96] considers an even simpler case in which the abstraction does not concern the variables on which the property ψ depends. Consequently, this is the case in which $\psi^\alpha = \psi$.

A more elaborate study in [DGG97] considers a more complex specification language - L_μ, which is a positive version of the μ-calculus.

None of these three articles considers explicitly the question of fairness requirements and how they are affected by the abstraction process.

Approaches based on simulation and studies of the properties they preserve are considered in [LGS$^+$95].

A linear-time application of abstract interpretation is proposed in [BBM95], applying the abstractions directly to the computational model of *fair transition systems* (FTS) which is very close to the FDS model considered here. However, the method is only applied for the verification of safety properties. Liveness, and therefore fairness, are not considered.

In [MP91a], a deductive methodology for proving temporal properties over infinite state system is presented. This methodology is based on a set of proof rules, each devised for a class of temporal formulas. In each of these rules, the proof of the temporal property is reduced to the proof of a (finite set of) first-order premises. This methodology is proved to be complete, relative to the underlying assertion language.

Both proof rules and their completeness are based on the FTS computation model [MP91b]. The translation of both rules and completeness proof to the fair discrete system (FDS) model used in this paper, is presented in [KP98a].

Verification diagrams, presented in [MP94], provide a graphical representation of the deductive proof rules, summarizing the necessary verification conditions. A verification diagram (VD) is a finite graph, which can be viewed as a finite abstraction of the verified system, with respect to the verified property.

In [BMS95, MBSU98], the notion of a verification diagram is generalized, allowing a uniform verification of arbitrary temporal formulas. The GVD (generalized verification diagram) can be viewed as an abstraction of the verified system which is justified deductively and verified by model checking. The GVD method is also shown to be sound and complete. The abstraction constructed by this method is based on the FTS computation model, and can be viewed as an ω-automaton with either Street ([BMS95]) or Muller ([MBSU98]) acceptance condition.

A dual method to VD and GVD is the deductive model checking (DMC) presented in [SUM99]. Similar to VD and GVD, this method tries to verify a temporal property φ over an infinite state system, using a finite graph representation. The procedure starts with the temporal tableau for the negated property ($\neg\varphi$), which is repeatedly refined until either a counter example is found or it is proved that a counter example can not exist. The paper presents a constructive method which, for infinite state systems, is not guaranteed to terminate. The method is shown to be complete, relative to the underlying assertion language, for proving general temporal properties.

An (LTL-based) general approach, similar to our VFA method, has been independently developed in [Uri99]. The claim of completeness there relies on the (relative) completeness established within [SUM99].

An important development in the theory and implementation of verification by finitary (and other types of) abstraction is reported in [BLO98a]. The paper describes the support system InVest [BLO98b], which employs various heuristics for the automatic generation of finitary abstractions for a given system, attempting to be precise (a concept introduced in Section 6) with respect to the atomic formulas appearing in the system as well as in the specification. For example, InVest has managed to compute automatically most of the abstractions presented in our examples such as Fig. 8 and Fig. 11.

2 A Computational Model: Fair Discrete Systems

As a computational model for reactive systems, we take the model of a *fair discrete system* (FDS), which is a slight variation on the model of *fair transition system* (FTS) [MP95]. The FDS model was first introduced in [KPR98] under the name "Fair Kripke Structure". The main difference between the FDS and FTS models is in the representation of fairness constraints. The advantage of the new representation is that it enables a unified representation of fairness constraints arising from both the system being verified, and the temporal property.

An FDS $\mathcal{D} : \langle V, \Theta, \rho, \mathcal{J}, \mathcal{C} \rangle$ consists of the following components.

- $V = \{u_1, ..., u_n\}$: A finite set of typed *system variables*, containing data and control variables. The set of *states* (interpretation) over V is denoted by Σ. Note that Σ can be both finite or infinite, depending on the domains of V.

- Θ : The *initial condition* – an *assertion* (first-order state formula) characterizing the initial states.

- ρ : A *transition relation* – an assertion $\rho(V, V')$, relating the values V of the variables in state $s \in \Sigma$ to the values V' in a $\mathcal{D}$-successor state $s' \in \Sigma$.

- $\mathcal{J} = \{J_1, \ldots, J_k\}$: A set of *justice* requirements (also called *weak fairness* requirements). The justice requirement $J \in \mathcal{J}$ is an assertion, intended to guarantee that every computation contains infinitely many J-states (states satisfying J).

- $\mathcal{C} = \{\langle p_1, q_1\rangle, \ldots \langle p_n, q_n\rangle\}$: A set of *compassion* requirements (also called *strong fairness* requirements). The compassion requirement $\langle p, q\rangle \in \mathcal{C}$ is a pair of assertions, intended to guarantee that every computation containing infinitely many p-states also contains infinitely many q-states.

We require that every state $s \in \Sigma$ has at least one $\mathcal{D}$-successor. This is often ensured by including in ρ the *idling* disjunct $V = V'$ (also called the *stuttering* step). In such cases, every state s is its own $\mathcal{D}$-successor.

Let $\mathcal{D}$ be an FDS for which the above components have been identified. We define a *computation* of $\mathcal{D}$ to be an infinite sequence of states $\sigma : s_0, s_1, s_2, ...$, satisfying the following requirements:

- *Initiality:* s_0 is initial, i.e., $s_0 \models \Theta$.
- *Consecution:* For each $j = 0, 1, ...$, the state s_{j+1} is a $\mathcal{D}$-successor of the state s_j.
- *Justice:* For each $J \in \mathcal{J}$, σ contains infinitely many J-positions
- *Compassion:* For each $\langle p, q\rangle \in \mathcal{C}$, if σ contains infinitely many p-positions, it must also contain infinitely many q-positions.

For an FDS $\mathcal{D}$, we denote by $\mathcal{C}omp(\mathcal{D})$ the set of all computations of $\mathcal{D}$. An FDS $\mathcal{D}$ is called *feasible* if $\mathcal{C}omp(\mathcal{D}) \neq \emptyset$, namely, if $\mathcal{D}$ has at least one computation. The feasibility of a finite-state FDS can be checked algorithmically, using symbolic model checking methods, as presented in [KPR98]. A state s is called $\mathcal{D}$-*accessible* if it appears in some computation of $\mathcal{D}$.

A finite or infinite state sequence σ is called a *run* of $\mathcal{D}$ if it satisfies the requirements of initiality and consecution but not, necessarily, any of the fairness requirements. System $\mathcal{D}$ is said to be *viable* if every finite run can be extended into a computation. One of the differences between the model of fair transition systems and the FDS model is that every FTS is viable by construction, while it is easy to define an FDS which is not viable, e.g., by having the justice set include the assertion *false*. On the other hand, every FDS which is derived from a program is viable.

Let $U \subseteq V$ be a set of variables. Let σ be an infinite sequence of states. We say the σ' is a U-variant of σ, if σ' agrees with σ on the interpretation of all variables in $V - U$, and disagree with σ only on the interpretation of variables in U. Similarly, we denote by $\sigma {\Downarrow}_U$ the *projection* of σ onto the subset U. That is, $\sigma {\Downarrow}_U$ is the sequence of U-states obtained by removing from the states of σ the valuation of the variables which belong to $V - U$. For the set of computations $\mathcal{C}omp(\mathcal{D})$ of and FDS $\mathcal{D}$, we denote by $\mathcal{C}omp(\mathcal{D}) {\Downarrow}_U$ the set of computations projected onto the set of variables U. Let $\mathcal{D}$ and $\mathcal{D}'$ be two FDS's. We denote by $\mathcal{C}omp(\mathcal{D}) {\Downarrow}_{\mathcal{D}'}$ the set of computations of $\mathcal{D}$ projected onto $V_{\mathcal{D}'}$, the set of variables of $\mathcal{D}'$.

All our concrete examples are given in SPL (Simple Programming Language), which is used to represent concurrent programs (e.g., [MP95], [MAB$^+$94]). Every SPL program can be compiled into an FDS in a straightforward manner (see [KPR98]). In particular, every statement in an SPL program contributes a disjunct to the transition relation. For example, the assignment statement

$$\ell_0 : y := x + 1;\ \ell_1 :$$

can be executed when control is at location ℓ_0. When executed, it assigns $x+1$ to y while control moves from ℓ_0 to ℓ_1. This statement contributes to ρ the disjunct

$$\rho_{\ell_0}: \quad at\_\ell_0 \ \wedge\ at\_\ell_1' \ \wedge\ y' = x + 1 \ \wedge\ x' = x.$$

The predicates at_ℓ_0 and at_ℓ_1' stand, respectively, for the assertions $\pi_i = 0$ and $\pi_i' = 1$, where π_i is the control variable denoting the current location within the process to which the statement belongs.

2.1 Synchronous Parallel Composition

Let $\mathcal{D}_1 : \langle V_1, \Theta_1, \rho_1, \mathcal{J}_1, \mathcal{C}_1 \rangle$ and $\mathcal{D}_2 : \langle V_2, \Theta_2, \rho_2, \mathcal{J}_2, \mathcal{C}_2 \rangle$ be two fair discrete systems. We define the *synchronous parallel composition* of $\mathcal{D}_1$ and $\mathcal{D}_2$, denoted by $\mathcal{D}_1 \,|||\, \mathcal{D}_2$, to be the system $\mathcal{D} = \langle V, \Theta, \rho, \mathcal{J}, \mathcal{C} \rangle$, where,

$$\begin{array}{rclcrcl} V & = & V_1 \cup V_2 & \qquad & \Theta & = & \Theta_1 \wedge \Theta_2 \\ \mathcal{J} & = & \mathcal{J}_1 \cup \mathcal{J}_2 & & \mathcal{C} & = & \mathcal{C}_1 \cup \mathcal{C}_2 \\ \rho & = & \rho_1 \ \wedge\ \rho_2 & & & & \end{array}$$

As implied by the definition, each of the basic actions of system $\mathcal{D}$ consists of the joint execution of an action of $\mathcal{D}_1$ and an action of $\mathcal{D}_2$. Thus, we can view the execution of $\mathcal{D}$ as the *joint execution* of $\mathcal{D}_1$ and $\mathcal{D}_2$.

The main, well established, use of the synchronous parallel composition is for coupling a system with a *tester* which tests for the satisfaction of a temporal formula, and then checking the feasibility of the combined system, as will be shown in the following sections. In this work, synchronous composition is also used for coupling the system with a *monitor*, used to ensure completeness of the data abstraction methodology presented in the following sections. We remind the reader that the concurrent composition of several SPL processes is an *asynchronous* composition based on interleaving.

2.2 From JDS to BDS

An FDS with no compassion requirements is called a *just discrete system* (JDS). A JDS with a single justice requirement is called a *Büchi discrete system* (BDS).

Let $\mathcal{D} : \langle V, \Theta, \rho, \mathcal{J}, \mathcal{C} : \emptyset \rangle$ be a JDS such that $\mathcal{J} = \{J_1, \ldots, J_k\}$ and $k > 1$. Let $\mathcal{B}$ be a BDS and $U = V_{\mathcal{D}} \cap V_{\mathcal{B}}$. We say that $\mathcal{D}$ is U-equivalent to the BDS $\mathcal{B}$, denoted $\mathcal{D} \sim_U \mathcal{B}$ iff $\mathcal{C}omp(\mathcal{D}) \Downarrow_U = \mathcal{C}omp(\mathcal{B}) \Downarrow_U$.

We define a BDS $\mathcal{B} : \langle V_{\mathcal{B}}, \Theta_{\mathcal{B}}, \rho_{\mathcal{B}}, \mathcal{J}_{\mathcal{B}} : \{J\}, \mathcal{C}_{\mathcal{B}} : \emptyset \rangle$ which is U-equivalent to $\mathcal{D}$ as follows:

- $V_{\mathcal{B}} = V \cup \{u\}$, where u is a new variable not in V, interpreted over the domain $[0..k]$.

- $\Theta_{\mathcal{B}} : \quad u = 0 \ \wedge\ \Theta.$

- $\rho_{\mathcal{B}} \;:\; \rho(V,V') \;\wedge\; \bigvee_{i=0}^{k}(u=i) \;\wedge\; u' = \left[\begin{array}{lcr} \textbf{case} & & \\ \;u=0 & : & 1\;; \\ \;u>0 \;\wedge\; J_i & : & (u+1)\,mod\;(k+1)\;; \\ \;true & : & u\;; \\ \textbf{esac} & & \end{array}\right]$

- $\mathcal{J}_{\mathcal{B}} = \{J\}$, where J is the single justice requirement $J\!:(u=0)$.

The transformation of a JDS to a BDS follows the transformation of generalized Büchi automata to Büchi automata [Cho74].

3 Requirement Specification Language: Temporal Logic

As a requirement specification language for reactive systems we take *linear temporal logic* (LTL) [MP91b]. For simplicity, we consider only the future fragment of the logic. Extending the approach to the full logic is straightforward.

We assume an underlying assertion language $\mathcal{L}$ which contains the predicate calculus augmented with fix-point operators.[1] We assume that $\mathcal{L}$ contains interpreted symbols for expressing the standard operations and relations over some concrete domains, such as the integers.

A *temporal formula* is constructed out of state formulas (assertions) to which we apply the boolean operators $\neg$ and $\vee$ (the other boolean operators can be defined from these), and the basic temporal operators $\bigcirc$ (*next*) and $\mathcal{U}$ (*until*).

A *model* for a temporal formula p is an infinite sequence of states $\sigma : s_0, s_1, ...$, where each state s_j provides an interpretation for the variables mentioned in p.

Given a model σ, we present an inductive definition for the notion of a temporal formula p holding at a position $j \geq 0$ in σ, denoted by $(\sigma, j) \models p$.

- For a state formula p, $\quad (\sigma, j) \models p \iff s_j \models p$
  That is, we evaluate p locally, using the interpretation given by s_j.
- $(\sigma, j) \models \neg p \iff (\sigma, j) \not\models p$
- $(\sigma, j) \models p \vee q \iff (\sigma, j) \models p$ or $(\sigma, j) \models q$
- $(\sigma, j) \models \bigcirc p \iff (\sigma, j+1) \models p$
- $(\sigma, j) \models p\mathcal{U}q \iff$ for some $k \geq j, (\sigma, k) \models q$,
  and for every i such that $j \leq i < k, (\sigma, i) \models p$

Additional temporal operators can be defined by $\diamondsuit\, p = true\mathcal{U}p$ (*eventually*) and $\square\, p = \neg \diamondsuit \neg p$ (*henceforth*).

For a temporal formula p and a position $j \geq 0$ such that $(\sigma, j) \models p$, we say that j is a *p-position* (in σ). If $(\sigma, 0) \models p$, we say that p *holds* on σ, and denote it by $\sigma \models p$. A formula p is called *satisfiable* if it holds on some model. A formula p is called *valid*, denoted by $\models p$, if it holds on all models. Two formulas p and q are defined to be *equivalent*,

[1] As is well known ([LPS81],) a first-order language is not adequate to express the assertions necessary for (relative) completeness of a proof system for proving validity of temporal properties of reactive programs. The use of minimal and maximal fix-points for relative completeness of the proof rules for liveness properties is discussed in [MP91a], based on [SdRG89]. However, the fix-points are not needed in the assertion language used to specify the components of an FDS (Θ, ρ, $\mathcal{J}$ and $\mathcal{C}$) or the set of its reachable states (see section 5).

denoted $p \sim q$, if $p \leftrightarrow q$ is valid, i.e $\sigma \models p$ iff $\sigma \models q$, for all models σ. We say that p and q are *congruent*, denoted $p \approx q$, if $\Box(p \leftrightarrow q)$ is valid, i.e. $(\sigma, j) \models p$ iff $(\sigma, j) \models q$ for all models σ and $j \geq 0$.

Note, that a state formula p is valid iff it holds at position 0 of all models. Our treatment here differs from [MP91b] in that we do not require the separate concept of state-validity.

Given an FDS $\mathcal{D}$ and a temporal formula p, we say that p is *$\mathcal{D}$-valid*, denoted by $\mathcal{D} \models p$, if p holds on all models which are computations of $\mathcal{D}$. In case the formula p contains auxiliary variables U which are not among the system variables of $\mathcal{D}$, we apply this definition to the extended system $\mathcal{D}_U$ obtained by adding U to the system variables of $\mathcal{D}$. Note, that the values of these variables in a computation of $\mathcal{D}_U$ are completely unconstrained since neither Θ_U nor ρ_U refer to them.

Let p be a temporal formula. We define the *vocabulary of* p as the set of all free variables in maximal state-subformulas of p. We say that p is *finitary* if the vocabulary V of p is finite and, for each variable $v \in V$, v ranges over a finite domain.

A temporal formula φ is called *relevant* for an FDS $\mathcal{D}$, if the only free variables appearing in φ are system variables of $\mathcal{D}$. Obviously, a formula purporting to describe a property of $\mathcal{D}$ can only refer (freely) to the system variables of $\mathcal{D}$.

4 Testers for Temporal Formulas

In this section, we present the construction of a *tester* for an LTL formula φ, which is a BDS T_φ characterizing all the sequences which satisfy φ. The construction of a temporal tester proceeds in two steps. In Subsection 4.1, we present a construction of a *pre-tester* Π_φ which is a JDS whose computations are all the sequences satisfying the formula φ. Then, in Subsection 4.3, we complete the construction by applying the transformation described in Subsection 2.2 which transforms the JDS Π_φ to a BDS T_φ which is the tester for the formula φ.

The notion of a temporal tester was first introduced in [KPR98], which was strongly inspired by [CGH94]. However, the construction in [KPR98] stopped at the level that we describe here as a pre-tester, and did not proceed to bring the system into a BDS form.

4.1 Pre-Testers

For a formula ψ, we write $\psi \in \varphi$ to denote that ψ is a sub-formula of (possibly equal to) φ. Formula ψ is called *principally temporal* if its main operator is a temporal operator.

The JDS Π_φ is given by

$$\Pi_\varphi: \quad \left\langle V_\varphi,\, \Theta_\varphi,\, \rho_\varphi, \mathcal{J}_\varphi, \mathcal{C}_\varphi : \emptyset \right\rangle,$$

where the components are specified as follows:

System Variables

The system variables of Π_φ consist of the vocabulary of φ plus a set of auxiliary boolean variables

$$X_\varphi: \quad \{x_p \mid p \in \varphi \text{ a principally temporal sub-formula of } \varphi\},$$

which includes an auxiliary variable x_p for every p, a principally temporal sub-formula of φ. The auxiliary variable x_p is intended to be true in a state of a computation iff the temporal formula p holds at that state.

We define a mapping χ which maps every sub-formula of φ into an assertion over V_φ.

$$\chi(\psi) = \begin{cases} \psi & \text{for } \psi \text{ a state formula} \\ \neg\chi(p) & \text{for } \psi = \neg p \\ \chi(p) \vee \chi(q) & \text{for } \psi = p \vee q \\ x_\psi & \text{for } \psi \text{ a principally temporal formula} \end{cases}$$

The mapping χ distributes over all boolean operators. When applied to a state formula it yields the formula itself. When applied to a principally temporal sub-formula p it yields the variable x_p.

Initial Condition

The initial condition of Π_φ is given by

$$\Theta_\varphi: \quad \chi(\varphi).$$

Thus, the initial condition requires that all initial states satisfy $\chi(\varphi)$.

Transition Relation

The transition relation of Π_φ is given by

$$\rho_\varphi: \quad \left\{ \bigwedge_{\bigcirc p \in \varphi} (x_{\bigcirc p} \ \leftrightarrow \ \chi'(p)) \ \wedge \ \bigwedge_{p\mathcal{U}q \in \varphi} (x_{p\mathcal{U}q} \ \leftrightarrow \ (\chi(q) \ \vee \ (\chi(p) \wedge x'_{p\mathcal{U}q}))) \right\}$$

Note that we use the form x_ψ when we know that ψ is principally temporal and the form $\chi(\psi)$ in all other cases. The expression $\chi'(\psi)$ denotes the primed version of $\chi(p)$. The conjunct of the transition relation corresponding to the *Until* operator is based on the following expansion formula:

$$p\mathcal{U}q \quad \Longleftrightarrow \quad q \vee (p \wedge \bigcirc(p\mathcal{U}q))$$

Fairness Requirements

The justice set of Π_φ is given by

$$\mathcal{J}_\varphi: \quad \{\chi(q) \vee \neg x_{p\mathcal{U}q} \mid p\mathcal{U}q \in \varphi\}.$$

Thus, we include in $\mathcal{J}_\varphi$ the disjunction $\chi(q) \vee \neg x_{p\mathcal{U}q}$ for every *until* formula $p\mathcal{U}q$ which is a sub-formula of φ. The justice requirement for the formula $p\mathcal{U}q$ ensures that the sequence contains infinitely many states at which $\chi(q)$ is true, or infinitely many states at which $x_{p\mathcal{U}q}$ is false.

The compassion set of Π_φ is always empty.

4.2 Correctness of the Construction

For a set of variables U, we say that sequence $\widetilde{\sigma}$ is a *U-variant* of sequence σ if σ and $\widetilde{\sigma}$ agree on the interpretation of all variables, except possibly the variables in U.

The following claim states that the construction of the tester Π_φ correctly captures the set of sequences satisfying the formula φ.

Claim 1 *A state sequence σ satisfies the temporal formula φ iff σ is an X_φ-variant of a computation of Π_φ.*

Proof (sketch)**:** Obviously, a tester is nothing more than a symbolic version of the construction of a temporal tableau (e.g. see [MW84], [LP85], [MP95]). Therefore, most of the necessary justification of Claim 1 can be taken from these papers.

Here, we would only like to elaborate on the salient point of the symbolic representation of the tester as consisting of several boolean variables, each representing one of the principally temporal sub-formulas. The general proof proceeds by induction on the size of the sub-formula, and we consider the crucial step of handling sub-formulas of the form $p\mathcal{U}q$ where, for simplicity, we assume that p and q are state formulas.

For such a formula, the pre-tester Π_φ contains a variable $x_{p\mathcal{U}q}$, the transition relation contains a conjunct $x_{p\mathcal{U}q} \iff q \lor (p \land x'_{p\mathcal{U}q})$, and the justice set contains a justice requirement $q \lor \neg x_{p\mathcal{U}q}$. We would like to show that, for every σ a computation of Π_φ and every position $j \geq 0$,

$$(\sigma, j) \models x_{p\mathcal{U}q} \quad \iff \quad (\sigma, j) \models p\mathcal{U}q. \tag{1}$$

Consider first the case that $(\sigma, j) \models p\mathcal{U}q$. By definition of the *until* operator, there exists a $k \geq j$ such that q holds at k, and p holds at all intermediate positions i, $j \leq i < k$. By the transition relation for $x_{p\mathcal{U}q}$, we can work down from k and establish that $x_{p\mathcal{U}q}$ holds at all positions $i = k, k-1, \ldots, j$.

In the other direction, assume that $(\sigma, j) \models x_{p\mathcal{U}q}$. Applying the transition relation to positions $j, j+1, \ldots,$ will result in one of two possibilities. Either q holds at some positions $k \geq j$ and p holds at all intermediate positions i, $j \leq i < k$, or $x_{p\mathcal{U}q}$, p, and $\neg q$ hold at all positions $i \geq j$. The first possibility yields $(\sigma, j) \models p\mathcal{U}q$, while the second possibility is ruled out by the justice requirement $q \lor \neg x_{p\mathcal{U}q}$ which is required to hold at infinitely many positions, including at least one beyond j. ▟

4.3 The Final Step: Transforming into a BDS

In the second step of the tester construction, we transform the JDS Π_φ into a BDS T_φ, using the JDS→BDS transformation presented in Subsection 2.2.

4.4 Additional Temporal Operators and an Example

The construction of pre-testers, as presented in the previous subsection, considered $\mathcal{U}$ and $\bigcirc$ as the only temporal operators. In most applications, we encounter formulas with the additional temporal operators $\Box$ and $\Diamond$. Obviously, these operators can be defined in terms of $\mathcal{U}$. However, it is very convenient to add to the construction a direct treatment of sub-formulas of the form $\Box\, p$ and $\Diamond\, p$.

This can be done as follows:

For every $\Diamond\, p$, a sub-formula of φ, add to X_φ the variable $x_{\Diamond p}$, and add to ρ_φ the conjunct:

$$x_{\Diamond p} \quad \Longleftrightarrow \quad \chi(p) \ \lor \ x'_{\Diamond p}$$

Also add to $\mathcal{J}_\varphi$ the justice requirement:

$$\chi(p) \ \lor \ \neg x_{\Diamond p}$$

For every $\Box\, p$, a sub-formula of φ, add to X_φ the variable $x_{\Box p}$, and add to ρ_φ the conjunct:

$$x_{\Box p} \quad \Longleftrightarrow \quad \chi(p) \ \land \ x'_{\Box p}$$

Also add to $\mathcal{J}_\varphi$ the justice requirement:

$$\neg\chi(p) \ \lor \quad x_{\Box p}$$

An Example

We conclude this section by an example of a tester constructed for the temporal formula

$$\varphi : \quad \Diamond\, \Box (x < 0) \ \land \ \neg \Diamond (at\_\ell_3),$$

where x and ℓ_3 refer to an example program which we will consider in the following sections.

Following the recipe presented in this section, the temporal tester T_φ is given by:

$$\begin{array}{rl}
V : & \pi, x : \textbf{natural}, f_1, g_2, f_3 : \textbf{boolean}, u : [0..3] \\
\Theta_\varphi : & f_1 \ \land \ \neg f_3 \\
\rho_\varphi : & \left(\begin{array}{ccccc}
f_1 & \leftrightarrow & g_2 & \lor & f_1' \quad \land \\
g_2 & \leftrightarrow & x < 0 & \land & g_2' \quad \land \\
f_3 & \leftrightarrow & at\_\ell_3 & \lor & f_3' \quad \land \\
u' = & \multicolumn{4}{l}{\left[\begin{array}{ll}
\textbf{case} & \\
\quad u = 0 & : 1 \ ; \\
\quad u = 1 \ \land \ (g_2 \ \lor \ \neg f_1) & : 2 \ ; \\
\quad u = 2 \ \land \ (x \geq 0 \ \lor \ g_2) & : 3 \ ; \\
\quad u = 3 \ \land \ (at\_\ell_3 \ \lor \ \neg f_3) & : 0 \ ; \\
\quad \mathit{true} & : u \ ; \\
\textbf{esac} &
\end{array}\right]}
\end{array}\right) \\
J_\varphi : & u = 0
\end{array}$$

For easier reference, we have renamed the variables of X_φ, letting f_1, g_2, and f_3 stand, respectively, for $x_{\Diamond \Box (x<0)}$, $x_{\Box (x<0)}$, and $x_{\Diamond\, at\_\ell_3}$. Note that the system variables for this tester includes π the program counter of the program for which the property $\Diamond\, \Box (x < 0) \ \land \ \neg \Diamond (at\_\ell_3)$ is claimed, and the natural variable x, which is also one of the program variables. The predicate at_ℓ_3 stands for the state formula $\pi = 3$.

4.5 The Testers T_φ, $T_{\neg\varphi}$, and T^{φ}_{true}

It is a known fact that the temporal tableaux of T_φ and $T_{\neg\varphi}$ have identical structure and fairness requirements and only differ in their initial states and conditions. This is also true of testers. The testers T_φ and $T_{\neg\varphi}$ have identical system variables, identical transition relation, and identical justice requirements. They only differ in their initial conditions which are $\Theta_\varphi = \chi(\varphi) \ \wedge \ (u = 0)$ for T_φ and $\Theta_{\neg\varphi} = \chi(\neg\varphi) \ \wedge \ (u = 0)$ for $T_{\neg\varphi}$.

We can thus view $T_{\neg\varphi}$ as obtained from T_φ by replacing the initial condition Θ by $\chi(\neg\varphi) \ \wedge \ (u = 0)$. Another variant of T_φ is $T^{\varphi}_{true} = \langle V_\varphi, \ (u = 0), \ \rho_\varphi, \ \mathcal{J}_\varphi, \ \mathcal{C}_\varphi : \emptyset\rangle$, which can be obtained from T_φ by replacing Θ, by $(true \ \ \wedge \ u = 0)$.

In an analogy to Claim 1, we can make the following statement, characterizing the sequences accepted by T^{φ}_{true}:

Every state sequence σ is an X_φ-variant of a computation of T^{φ}_{true}.

This claim states that, modulo renaming of the internal variables, every sequence is accepted by (is a computation of) T^{φ}_{true}.

5 Verifying Infeasibility of Büchi Discrete Systems

In the following, we present a general proof method for establishing that a BDS is infeasible.

A *well-founded domain* $(\mathcal{W}, \prec)$ consists of a set $\mathcal{W}$ and a total ordering relation $\prec$ over $\mathcal{W}$ such that there does not exist an infinitely descending sequence, i.e., a sequence of the form

$$a_0 \ \succ \ a_1 \ \succ \ a_2 \ \succ \ \cdots \ ,$$

A *ranking function* for an FDS $\mathcal{D}$ is a function δ mapping the states of $\mathcal{D}$ into a well-founded domain.

The standard approach to prove infeasibility of a BDS $\mathcal{B}\colon\langle V, \Theta, \rho, \mathcal{J} : \{J\}, \mathcal{C} : \emptyset\rangle$, is to define a ranking function δ which maps the reachable states of $\mathcal{B}$ into a well founded domain. The ranking function is required to satisfy the conditions that every transition of $\mathcal{B}$ does not increase the rank and every transition into a state satisfying J, the single justice requirement of $\mathcal{B}$, decreases the rank. The (possibly infinite) set of reachable states of $\mathcal{B}$ can be characterized (over-approximated) by an inductive assertion φ. The infeasibility of $\mathcal{B}$ can then be derived from the rule WELL, presented in Fig. 1.
Rule WELL is both sound and (relatively) complete. Soundness of the rule means that, given a BDS $\mathcal{B}$, if we can find a ranking function δ and an assertion φ, such that φ and δ satisfy the three premises W1–W3, then $\mathcal{B}$ is indeed infeasible. To see this, assume, to the contrary, that $\mathcal{B}$ is feasible. Then $\mathcal{B}$ has an infinite computation $\sigma\colon s_0, s_1, \ldots$, such that $s_i \models J$ for infinitely many states s_i in σ. Then, from premises W2 and W3, there exists an infinite sequence of states over which the ranking function δ decreases infinitely many times, and never increases. Since δ is defined over a well-founded domain, this is clearly impossible, contradicting our assumption.

The completeness of rule WELL is stated by the following claim:

Claim 2 *Let $\mathcal{B}\colon\langle V, \Theta, \rho, \mathcal{J} : \{J\}, \mathcal{C} : \emptyset\rangle$ be a* BDS*. If $\mathcal{B}$ is infeasible, then there exist an assertion φ, a well founded domain $(\mathcal{W}, \prec)$ and a ranking function $\delta\colon\Sigma_V \mapsto \mathcal{W}$ satisfying the premises of rule* WELL.

For an assertion φ,
a well founded domain $(\mathcal{W}, \prec)$,
and a ranking function $\delta : \Sigma_V \mapsto \mathcal{W}$

$$\begin{array}{lllll} \text{W1.} & \Theta & \rightarrow & \varphi \\ \text{W2.} & \rho \wedge \varphi & \rightarrow & \varphi' \wedge \delta' \preceq \delta \\ \text{W3.} & \rho \wedge \varphi \wedge J' & \rightarrow & \varphi' \wedge \delta' \prec \delta \\ \hline & & \mathcal{C}omp(\mathcal{B}) = \emptyset & & \end{array}$$

Figure 1: Rule WELL.

Proof (sketch)**:** To prove the claim, we have to find both an assertion φ and a ranking function δ which satisfy the premises W1–W3 of rule WELL.

The proof of existence of an assertion φ characterizing the set of all reachable states of a BDS is presented in [MP91a] and discussed in more detail in [MP91b] (Section 2.5). The assertion (using predicate calculus) is constructed as an encoding of the finite path to a reachable state, using the initial condition Θ and the transition relation ρ of $\mathcal{B}$ to constrain the path.

The existence of a a well founded domain $(\mathcal{W}, \prec)$ and ranking function δ satisfying the premises W1–W3, is shown in [Var91], based on [LPS81]. The syntactic representation of a well founded ranking using an assertion language based on the predicate calculus augmented with minimal and maximal fixpoints operators, is discussed in [MP91a] based on [SdRG89].

6 Verification by Finitary Abstraction

In this section, we present a general methodology for *data abstraction*, strongly inspired by the notion of abstract interpretation [CC77]. Let $\mathcal{D} = \langle V, \Theta, \rho, \mathcal{J}, \mathcal{C} \rangle$ be an FDS, and Σ denote the set of states of $\mathcal{D}$, the *concrete states*. Let $\alpha : \Sigma \mapsto \Sigma_A$ be a mapping of concrete states into *abstract states*. We say that α is a finitary abstraction mapping, if Σ_A is a finite set. The strategy of *verification by finitary abstraction* (VFA) can be summarized as follows:

- Define a finitary abstraction mapping α to abstract the (possibly infinite) *concrete* FDS $\mathcal{D}$ into a *finite*, *abstract* FDS $\mathcal{D}^\alpha$.
- Abstract the *concrete* temporal property ψ into a *finitary abstract* temporal property ψ^α.
- Verify $\mathcal{D}^\alpha \models \psi^\alpha$.
- Infer $\mathcal{D} \models \psi$.

An implementation of this general strategy which specifies a recipe for defining the abstractions $\mathcal{D}^\alpha$ and ψ^α for a given α is called an *abstraction method.*

An abstraction method is said to be *safe* (equivalently, *sound*) if, for every FDS $\mathcal{D}$, temporal formula ψ, and a state abstraction mapping α (not necessarily finitary), $\models \psi^\alpha$ implies $\models \psi$, and $\mathcal{D}^\alpha \models \psi^\alpha$ implies $\mathcal{D} \models \psi$.

6.1 Safe Abstraction of Temporal Formulas

To provide a syntactic representation of the abstraction mapping, we assume a set of *abstract variables* V_A and a set of expressions $\mathcal{E}^\alpha$, such that the equality $V_A = \mathcal{E}^\alpha(V)$ syntactically represents the semantic mapping α.

Let $p(V)$ be an assertion. We wish to define the abstraction $p^\alpha(V_A)$ such that $\models p^\alpha(V_A)$ implies $\models p(V)$. We introduce the operator α^-, defined by

$$\alpha^-(p(V)):\quad \forall V\ \Big(V_A = \mathcal{E}^\alpha(V) \quad \rightarrow \quad p(V)\Big) \quad \wedge \quad map(V_A),$$

where $map(V_A) : \exists V\Big(V_A = \mathcal{E}^\alpha(V)\Big)$. Note that the free variables of $\alpha^-(p(V))$ are the abstract variables V_A. The assertion $\alpha^-(p)$ holds over an abstract state $S \in \Sigma_A$ iff S is *mappable* (is the α-image of some concrete state) and the assertion p holds over *all* concrete states $s \in \Sigma$ such that $s \in \alpha^{-1}(S)$. Alternatively, $\alpha^-(p)$ is the largest set of mappable states $X \subseteq \Sigma_A$ such that $\alpha^{-1}(X) \subseteq \|p\|$, where $\|p\|$ represents the set of states which satisfy the assertion p. If $\alpha^-(p)$ is valid, then $\|\alpha^-(p)\| = \Sigma_A$ implying $\alpha^{-1}(\|\alpha^-(p)\|) = \Sigma$ which, by the above inclusion, leads to $\|p\| = \Sigma$ establishing the validity of p.

For complex formulas, we have to consider assertions which are nested within an odd number of negations. To abstract an assertion under such a context, we define the operator α^+, as follows

$$\alpha^+(p(V)):\quad \exists V\ \Big(V_A = \mathcal{E}^\alpha(V) \quad \wedge \quad p(V)\Big).$$

The assertion $\alpha^+(p)$ holds over an abstract state $S \in \Sigma_A$ iff the assertion p holds over *some* concrete state $s \in \Sigma$ such that $s \in \alpha^{-1}(S)$, i.e., some state s such that $S = \alpha(s)$. Alternatively, $\alpha^+(p)$ is the smallest set $X \subseteq \Sigma_A$ such that $\|p\| \subseteq \alpha^{-1}(X)$.

Note the duality relations holding between α^+ and α^-, which can be expressed by the equivalences

$$\neg\alpha^+(p) \quad \sim \quad map(V_A) \rightarrow \alpha^-(\neg p) \tag{2}$$
$$\neg\alpha^-(p) \quad \sim \quad map(V_A) \rightarrow \alpha^+(\neg p) \tag{3}$$

or, equivalently, by

$$\alpha^+(\neg p) \quad \sim \quad \neg\alpha^-(p) \ \wedge \ map(V_A) \tag{4}$$
$$\alpha^-(\neg p) \quad \sim \quad \neg\alpha^+(p) \ \wedge \ map(V_A) \tag{5}$$

An abstraction α is said to be *precise with respect to an assertion* p if $\alpha^+(p) \sim \alpha^-(p)$. For such cases, we will sometimes write $\alpha^+(p)$ simply as $\alpha(p)$. As will be shown in Claim 9, a sufficient condition for α to be precise w.r.t. p is that the abstract variables include a boolean variable B_p with the α-definition $B_p = p$.

Having defined the abstractions α^- and α^+ which operate on assertions, we lift them to the abstractions α^-_τ and α^+_τ which can be applied to temporal formulas.

For a temporal formula φ and an occurrence p of a state sub-formula within φ, we say that p is a *maximal state sub-formula* (MSS) if it is not properly contained within another state sub-formula of φ.

The *universal* (or *contracting*) *abstraction* $\alpha^-_\tau(\varphi)$ is obtained by replacing

- Each MSS p occurring under a *positive polarity* (under an even number of negations) by $\alpha^-(p)$, and
- Each MSS p occurring under a *negative polarity* (under an odd number of negations) by $\alpha^+(p)$.

Similarly, the *existential* (or *expanding*) *abstraction* $\alpha^+_\tau(\varphi)$ is obtained by replacing

- Each MSS p occurring under a *positive polarity* (under an even number of negations) by $\alpha^+(p)$, and
- Each MSS p occurring under a *negative polarity* (under an odd number of negations) by $\alpha^-(p)$.

These definitions are equivalent to the following inductive definition:

For a state formula p,

$$\alpha^-_\tau(p) \;=\; \alpha^-(p) \qquad\qquad \alpha^+_\tau(p) \;=\; \alpha^+(p)$$

For a formula $\varphi \in \{\neg p, p \vee q, \bigcirc p, p\mathcal{U}q\}$, which is not a state formula,

$$\begin{array}{lcllcl}
\alpha^-_\tau(\neg p) & = & \neg\alpha^+_\tau(p) & \alpha^+_\tau(\neg p) & = & \alpha^-_\tau(p) \\
\alpha^-_\tau(p \vee q) & = & \alpha^-_\tau(p) \vee \alpha^-_\tau(q) & \alpha^+_\tau(p \vee q) & = & \alpha^+_\tau(p) \vee \alpha^+_\tau(q) \\
\alpha^-_\tau(\bigcirc p) & = & \bigcirc\, \alpha^-_\tau(p) & \alpha^+_\tau(\bigcirc p) & = & \bigcirc\, \alpha^+_\tau(p) \\
\alpha^-_\tau(p\mathcal{U}q) & = & (\alpha^-_\tau(p))\mathcal{U}(\alpha^-_\tau(q)) & \alpha^+_\tau(p\mathcal{U}q) & = & (\alpha^+_\tau(p))\mathcal{U}(\alpha^+_\tau(q))
\end{array}$$

Note that these definitions strongly depend on the syntactic representation of the temporal formula φ. In general, equivalent temporal formulas may have different abstractions. For example, the contracting abstractions of the equivalent formulas

$$p \vee (q \vee \diamondsuit\, r) \qquad \text{and} \qquad (p \vee q) \vee \diamondsuit\, r,$$

where p, q, and r are assertions (state formulas) are respectively given by the formulas

$$\alpha^-(p) \vee \alpha^-(q) \vee \diamondsuit\, \alpha^-(r) \qquad \text{and} \qquad \alpha^-(p \vee q) \vee \diamondsuit\, \alpha^-(r),$$

which may be inequivalent. Similarly, the respective abstractions of

$$p \wedge (q \wedge \square\, \mathrm{T}) \qquad \text{and} \qquad p \wedge q$$

are

$$\alpha^+(p) \wedge \alpha^+(q) \qquad \text{and} \qquad \alpha^+(p \wedge q).$$

A similar problem exists with formulas containing negation. For example, for the equivalent formulas $\varphi_1 : \neg p$ and $\varphi_2 : \neg(p \;\wedge\; \bigcirc\, \mathrm{T})$, we obtain

$$\begin{array}{lclcll}
\alpha^-_\tau(\varphi_1) & = & \alpha^-_\tau(\neg p) & = & & \alpha^-(\neg p) \\
\alpha^-_\tau(\varphi_2) & = & \alpha^-_\tau(\neg(p \;\wedge\; \bigcirc\, \mathrm{T})) & = & \neg(\alpha^+(p) \;\wedge\; \bigcirc\, \alpha^+(\mathrm{T})) \;\sim & \neg(\alpha^+(p))
\end{array}$$

Due to equivalence 2, these two abstractions are not in general equivalent.

Claim 3 *Let ψ be a temporal formula and α be an abstraction mapping. Then*

$$\models \alpha_\tau^-(\psi) \quad \textit{implies} \quad \models \psi$$

Proof: The proof is by induction on the structure of the formula. The induction hypotheses are given by:

For every state sequence $\sigma : s_0, s_1, \ldots$ and position $j \geq 0$,

$$(\alpha(\sigma), j) \models \alpha_\tau^-(\psi) \ \text{ implies } \ (\sigma, j) \models \psi, \qquad \text{and} \tag{6}$$

$$(\sigma, j) \models \psi \ \text{ implies } \ (\alpha(\sigma), j) \models \alpha_\tau^+(\psi) \tag{7}$$

where, $\alpha(\sigma) = \alpha(s_0), \alpha(s_1), \ldots$.

The base case is for ψ being a state formula. Let s_j be the state at position $j \geq 0$ of σ. Denote by $U^j = s_j[V]$ and $U_A^j = S_j[V_A]$, the values of the system variables V and V_A in the states s_j and $S_j = \alpha(s_j)$, respectively.

First, assume that $S_j \models \alpha_\tau^-(\psi)$, implying that $(\forall V \cdot U_A^j = \mathcal{E}^\alpha(V) \rightarrow \psi(V))$ evaluates to *true* over S_j. By substituting U^j for V and using the equality $U_A^j = \mathcal{E}^\alpha(U^j)$, we conclude that $\psi(U^j)$ evaluates to *true*. That is, $(\sigma, j) \models \psi$.

Next, assume $(\sigma, j) \models \psi$, namely $\psi(U^j)$ evaluates to true. Since $U_A^j = \mathcal{E}^\alpha(U^j)$, then $\exists V \cdot U_A^j = \mathcal{E}^\alpha(V) \wedge \psi(V)$, implying $(\alpha(\sigma), j) \models \alpha_\tau^+(\psi)$.

We proceed by considering the inductive step. Let p and q to be two temporal formulas satisfying the induction hypothesis. We have to show that each of the formulas $p \vee q$, $\neg p$, $\bigcirc p$, $p \mathcal{U} q$ satisfies the hypothesis. We show the proof for $\psi : p \vee q$ and $\psi : \neg p$. The proof for the other two formulas is similar.

Consider the formula $\psi : p \vee q$. Assume first that $(\alpha(\sigma), j) \models \alpha_\tau^-(p \vee q)$. From the definition of α_τ^-, we get $(\alpha(\sigma), j) \models \alpha_\tau^-(p) \vee \alpha_\tau^-(q)$. By the definition of satisfiability of temporal formulas, $(\alpha(\sigma), j) \models \alpha_\tau^-(p) \vee \alpha_\tau^-(q)$ implies $(\alpha(\sigma), j) \models \alpha_\tau^-(p)$ or $(\alpha(\sigma), j) \models \alpha_\tau^-(q)$ which, by the inductive hypothesis (6), implies $(\sigma, j) \models p$ or $(\sigma, j) \models q$. Finally, from the definition of satisfiability of temporal formulas, $(\sigma, j) \models p$ or $(\sigma, j) \models q$ implies $(\sigma, j) \models (p \vee q)$.

Next, assume $(\sigma, j) \models (p \vee q)$. Then, from the definition of temporal satisfiability, we conclude that $(\sigma, j) \models p$ or $(\sigma, j) \models q$. By the induction hypothesis (7), this implies $(\alpha(\sigma), j) \models \alpha_\tau^+(p)$ or $(\alpha(\sigma), j) \models \alpha_\tau^+(q)$, which, by the definition of temporal satisfiability, implies $(\alpha(\sigma), j) \models \alpha_\tau^+(p) \vee \alpha_\tau^+(q)$. Finally, from the definition of α_τ^+, we get $(\alpha(\sigma), j) \models \alpha_\tau^+(p \vee q)$.

Finally, we consider the formula $\psi : \neg p$, where p is not a state formula. Assume first that $(\alpha(\sigma), j) \models \alpha_\tau^-(\psi) = \alpha_\tau^-(\neg p)$. According to the definition, $\alpha_\tau^-(\neg p) = \neg \alpha_\tau^+(p)$. According to the definition of satisfiability of temporal formulas, $(\alpha(\sigma), j) \models \neg \alpha_\tau^+(p)$ implies $(\alpha(\sigma), j) \not\models \alpha_\tau^+(p)$. By the counter-positive of the induction hypothesis (7), $(\alpha(\sigma), j) \not\models \alpha_\tau^+(p)$ implies $(\sigma, j) \not\models p$, leading to $(\sigma, j) \models \neg p$. This establishes the induction hypothesis (6) for $\psi = \neg p$.

For Hypothesis (7), assume that $(\sigma, j) \models \psi$, i.e., $(\sigma, j) \models \neg p$. By the definition of temporal satisfiability, this implies $(\sigma, j) \not\models p$. By the counter-positive of Hypothesis (6) applied to p, we can conclude $(\alpha(\sigma), j) \not\models \alpha_\tau^-(p)$, leading to $(\alpha(\sigma), j) \models \neg \alpha_\tau^-(p)$ which, by the definition of $\alpha^+_\tau(p)$, leads to $(\alpha(\sigma), j) \models \alpha^+_\tau(\neg p)$. This establish the second clause of the induction hypothesis for $\neg p = \psi$.

To conclude the proof, we show that the inductive hypotheses implies the claim. Let $\sigma : s_0, s_1, \ldots$ be a (concrete) state sequence. We have to show that $\sigma \models \psi$. Let $\alpha(\sigma) = \alpha(s_0), \alpha(s_1) \ldots$. Since $\alpha(\sigma) \models \alpha^-_\tau(\psi)$ (left hand side of the claim), which is a shorthand for $\alpha(\sigma, 0) \models \alpha^-_\tau(\psi)$, it follows by Equation (6) that $(\sigma, 0) \models \psi$, which can be rewritten as $\sigma \models \psi$.

$\lrcorner$

In the following sections, we denote by ψ^α the contracting abstraction $\alpha^-_\tau(\psi)$ of the temporal formula ψ.

6.2 Safe Abstraction of FDS's

In the previous subsection, we established that the abstraction of the temporal formula ψ into $\psi^\alpha = \alpha^-_\tau(\psi)$ is *safe* (equivalently *sound*) in the sense that if ψ^α is valid, then so is ψ.

Here we will establish sufficient conditions for the joint abstraction of the FDS $\mathcal{D}$ and the temporal formula ψ to be safe (sound) in the sense that $\mathcal{D}^\alpha \models \psi^\alpha$ implies $\mathcal{D} \models \psi$. To do so, we reduce the problem of the safe joint abstraction of an FDS and a temporal property into the problem of safe abstraction of a single temporal property, a problem that has been solved in the preceding subsection.

Given an FDS $\mathcal{D} = \langle V, \Theta, \rho, \mathcal{J}, \mathcal{C}\rangle$, there exists a temporal formula $Sem(\mathcal{D})$, called the *temporal semantics* of $\mathcal{D}$ [Pnu81], such that, for every infinite state sequence σ, $\sigma \models Sem(\mathcal{D})$ *iff* $\sigma \in \mathcal{C}omp(\mathcal{D})$. The temporal semantics of an FDS $\mathcal{D}$ is given by

$$Sem(\mathcal{D}):\quad \Theta(V) \;\wedge\; \Box\,\rho(V, \bigcirc V) \;\wedge\; \bigwedge_{J\in\mathcal{J}} \Box\Diamond J(V) \;\wedge\; \bigwedge_{(p,q)\in\mathcal{C}} (\Box\Diamond p(V) \rightarrow \Box\Diamond q(V)),$$

where we use the temporal expression $\bigcirc V$ to denote the *next values* of the system variables V. Given a verification problem $\mathcal{D} \overset{?}{\models} \psi$, we construct the temporal formula

$$Ver(\mathcal{D}, \psi):\quad Sem(\mathcal{D}) \;\rightarrow\; \psi.$$

It is not difficult to establish that $\mathcal{D} \models \psi$ iff $Ver(\mathcal{D}, \psi)$ is valid.

Applying a safe α-abstraction to $Ver(\mathcal{D}, \psi)$, we obtain

$$\begin{array}{l}\alpha^-_\tau(Ver(\mathcal{D}, \psi)) = \\ \left(\alpha^+(\Theta) \;\wedge\; \Box\,\alpha^{++}(\rho)(V_A, \bigcirc V_A) \;\wedge\; \left(\begin{array}{l}\displaystyle\bigwedge_{J\in\mathcal{J}} \;\Box\Diamond\alpha^+(J) \;\;\wedge \\ \displaystyle\bigwedge_{(p,q)\in\mathcal{C}} (\Box\Diamond\alpha^-(p) \rightarrow \Box\Diamond\alpha^+(q)\end{array}\right)\right) \;\;\rightarrow\;\; \alpha^-_\tau(\psi).\end{array}$$

where

$$\alpha^{++}(\rho)(V_A, V'_A):\quad \exists V, V' \left(V_A = \mathcal{E}^\alpha(V) \;\;\wedge\;\; V'_A = \mathcal{E}^\alpha(V') \;\;\wedge\;\; \rho(V, V')\right).$$

Based on the way $\alpha_{\mathcal{T}}^{-}(\mathit{Ver}(\mathcal{D},\psi))$ abstracts the different components of $\mathcal{D}$, we define the *α-abstracted version* of $\mathcal{D}$ to be the FDS $\mathcal{D}^{\alpha}=\langle V_{A},\Theta^{\alpha},\rho^{\alpha},\mathcal{J}^{\alpha},\mathcal{C}^{\alpha}\rangle$, where

$$\begin{array}{rclrcl}\Theta^{\alpha} & = & \alpha^{+}(\Theta) & \rho^{\alpha} & = & \alpha^{++}(\rho)\\ \mathcal{J}^{\alpha} & = & \{\alpha^{+}(J)\mid J\in\mathcal{J}\} & \mathcal{C}^{\alpha} & = & \{(\alpha^{-}(p),\alpha^{+}(q))\mid(p,q)\in\mathcal{C}\}\end{array}$$

The following claim defines our VFA recipe and states its soundness (safety).

Claim 4 (Soundness) *The abstraction method which, for a given α, abstracts ψ into $\alpha_{\mathcal{T}}^{-}(\psi)$ and abstracts $\mathcal{D}$ into $\mathcal{D}^{\alpha}=\langle V_{A},\Theta^{\alpha},\rho^{\alpha},\mathcal{J}^{\alpha},\mathcal{C}^{\alpha}\rangle$, is safe. That is,*

$$\mathcal{D}^{\alpha}\models\psi^{\alpha}\qquad\textit{implies}\qquad\mathcal{D}\models\psi.$$

Proof: Assume that $\mathcal{D}^{\alpha}\models\psi^{\alpha}$, and show that $\mathcal{D}\models\psi$. Let $\sigma: s_0,s_1,\ldots$ be a computation of $\mathcal{D}$. We will show that $\sigma\models\psi$.

Consider the abstracted state sequence $\sigma_{A}: S_0,S_1,\ldots$, where $S_j=\alpha(s_j)$ for every $j\geq 0$. We will show that σ_{A} is a computation of $\mathcal{D}^{\alpha}$. Since $s_0\models\Theta$, we conclude by Equation (7) that $\alpha(s_0)\models\alpha^{+}(\Theta)$, implying $S_0\models\Theta^{\alpha}$. In a similar way, we conclude that $\langle s_j,s_{j+1}\rangle\models\rho$ implies $\langle S_j,S_{j+1}\rangle\models\alpha^{++}(\rho)$, leading to $\langle S_j,S_{j+1}\rangle\models\rho^{\alpha}$.

Next, consider the fulfillment of the justice requirements. For every $J\in\mathcal{J}$, we have that σ contains infinitely many positions $j\geq 0$ such that $s_j\models J$. By Equation (7), each of these positions satisfies $S_j\models\alpha^{+}(J)$, leading to the fact that σ_{A} fulfills each of the justice requirements in $\mathcal{J}^{\alpha}$.

Moving on to compassion, consider the compassion requirement $(p,q)\in\mathcal{C}$. Assume that σ_{A} contains infinitely many positions $j\geq 0$ such that $S_j\models\alpha^{-}(p)$. By Equation (6), each of these positions satisfies $s_j\models p$. Since σ is a computation of $\mathcal{D}$, satisfying all of its compassion requirements, σ must contain infinitely many positions $k\geq 0$ satisfying $s_k\models q$. By Equation (7), each of these positions satisfy $S_k\models\alpha^{+}(q)$. Consequently, σ_{A} fulfills the compassion requirement $(\alpha^{-}(p),\alpha^{+}(q))$. We conclude that σ_{A} is a computation of $\mathcal{D}^{\alpha}$.

Having assumed $\mathcal{D}^{\alpha}\models\psi^{\alpha}$, it follows that $\sigma_{A}\models\alpha_{\mathcal{T}}^{-}(\psi)$ which, by Equation (6), implies $\mathcal{D}\models\psi$. ▟

As an example, consider program BAKERY-2, presented in Fig. 2.
Program BAKERY-2 is obviously an infinite-state system, since the variables y_1 and y_2 can assume arbitrarily large values.

The temporal properties we wish to establish are given by

$$\begin{array}{lcl}\psi_{exc} & : & \Box\neg(at\_\ell_4\ \wedge\ at\_m_4)\\ \psi_{acc} & : & \Box(at\_\ell_2\quad\rightarrow\quad\Diamond\, at\_\ell_4),\end{array}$$

The safety property ψ_{exc} requires *mutual exclusion*, guaranteeing that the two processes never co-reside in their respective critical section at the same time. The liveness property ψ_{acc} requires *accessibility* for process P_1, guaranteeing that, whenever P_1 reaches location ℓ_2 it will eventually reach location ℓ_4.

Following [BBM95], we define abstract boolean variables $B_{p_1},B_{p_2},\ldots,B_{p_k}$, one for each atomic data formula, where the atomic data formulas for BAKERY-2 are $y_1=0$,

```
local  $y_1, y_2$  : natural where $y_1 = y_2 = 0$

$\ell_0$ : loop forever do
    $\ell_1$ : NonCritical
    $\ell_2$ : $y_1 := y_2 + 1$
    $\ell_3$ : await $y_2 = 0 \;\vee\; y_1 < y_2$
    $\ell_4$ : Critical
    $\ell_5$ : $y_1 := 0$

                    − $P_1$ −

||

$m_0$ : loop forever do
    $m_1$ : NonCritical
    $m_2$ : $y_2 := y_1 + 1$
    $m_3$ : await $y_1 = 0 \;\vee\; y_2 \leq y_1$
    $m_4$ : Critical
    $m_5$ : $y_2 := 0$

                    − $P_2$ −
```

Figure 2: Program BAKERY-2: the Bakery algorithm for two processes.

$y_2 = 0$, and $y_1 < y_2$. Note that the formula $y_2 \leq y_1$ is equivalent to the negation of $y_1 < y_2$ and needs not be included as an independent atomic formula.

The abstract system variables consist of the concrete control variables, which are left unchanged, and a set of abstract boolean variables $B_{p_1}, B_{p_2}, \ldots, B_{p_k}$. The abstraction mapping α is defined by

$$\alpha: \quad \{B_{p_1} = p_1, B_{p_2} = p_2, \ldots, B_{p_k} = p_k\}$$

That is, the boolean variable B_{p_i} has the value *true* in the abstract state iff the assertion p_i holds at the corresponding concrete state.

It is straightforward to compute the α-induced abstractions of the initial condition Θ^α and the transition relation ρ^α. In Fig. 3, we present program BAKERY-2 (with a capital B), the α-induced abstraction of program BAKERY-2.

```
local  $B_{y_1=0}, B_{y_2=0}, B_{y_1<y_2}$  : boolean initially $B_{y_1=0} = B_{y_2=0} = 1, B_{y_1<y_2} = 0$

$\ell_0$ : loop forever do
    $\ell_1$ : NonCritical
    $\ell_2$ : $(B_{y_1=0}, B_{y_1<y_2}) := (0, 0)$
    $\ell_3$ : await $B_{y_2=0} \;\vee\; B_{y_1<y_2}$
    $\ell_4$ : Critical
    $\ell_5$ : $(B_{y_1=0}, B_{y_1<y_2}) := (1, \neg B_{y_2=0})$

                    − $P_1$ −

||

$m_0$ : loop forever do
    $m_1$ : NonCritical
    $m_2$ : $(B_{y_2=0}, B_{y_1<y_2}) := (0, 1)$
    $m_3$ : await $B_{y_1=0} \;\vee\; \neg B_{y_1<y_2}$
    $m_4$ : Critical
    $m_5$ : $(B_{y_2=0}, B_{y_1<y_2}) := (1, 0)$

                    − $P_2$ −
```

Figure 3: Program BAKERY-2: the Bakery algorithm for two processes.

Since the properties we wish to verify refer only to the control variables (through the *at_ℓ* and *at_m* expressions), they are not affected by the abstraction. Program BAKERY-2 is a finite-state program, and we can apply model checking to verify that it satisfies the two properties of mutual exclusion and accessibility. By Claim 4, we can infer that the original program BAKERY-2 also satisfies these two temporal properties.

6.3 Properties of α^+ and α^{++}

It is straightforward to show that the assertion abstraction α^+ distributes over disjunction. That is, for every assertions p and q,

$$\alpha^+(p \lor q) \quad \sim \quad \alpha^+(p) \lor \alpha^+(q).$$

To see this, we recall the definition of α^+ and observe the following chain of equivalences:

$$\begin{array}{l} \alpha^+(p \lor q) \quad \sim \quad \exists V : V_A = \mathcal{E}^\alpha(V) \land (p(V) \lor q(V)) \qquad \sim \\ \exists V : (V_A = \mathcal{E}^\alpha(V) \land p(V)) \lor (V_A = \mathcal{E}^\alpha(V) \land q(V)) \qquad \sim \\ (\exists V : V_A = \mathcal{E}^\alpha(V) \land p(V)) \lor (\exists V : V_A = \mathcal{E}^\alpha(V) \land q(V)) \quad \sim \quad \alpha^+(p) \lor \alpha^+(q) \end{array}$$

On the other hand, α^+ does not distribute over conjunctions. For the general case, we can only claim that

$$\alpha^+(p \land q) \quad \text{implies} \quad \alpha^+(p) \land \alpha^+(q).$$

For the special case that α is precise with respect to q (i.e., $\alpha^+(q) \sim \alpha^-(q)$), we do have the equivalence

$$\alpha^+(p \land q) \quad \sim \quad \alpha^+(p) \land \alpha^+(q). \tag{8}$$

To see this, it is only necessary to establish that $\alpha^+(p) \land \alpha^+(q)$ implies $\alpha^+(p \land q)$. This is established by the following chain of equivalences/implications:

$$\begin{array}{l} \alpha^+(p) \land \alpha^+(q) \quad \sim \quad \alpha^+(p) \land \alpha^-(q) \quad \sim \\ \exists V : (V_A = \mathcal{E}^\alpha(V) \land p(V)) \quad \land \quad \forall V : (V_A = \mathcal{E}^\alpha(V) \rightarrow q(V)) \qquad \text{implies} \\ \exists V : V_A = \mathcal{E}^\alpha(V) \land (p(V) \land q(V)) \quad \sim \quad \alpha^+(p \land q) \end{array}$$

By symmetry, $\alpha^+(p \land q) \sim (\alpha^+(p) \land \alpha^+(q))$ also for the case that α is precise with respect to p. Similarly, we can establish that always $\alpha^-(p \land q)$ is equivalent to $\alpha^-(p) \land \alpha^-(q)$ and, under the assumption that α is precise with respect to q, also $\alpha^-(p \lor q)$ is equivalent to $\alpha^-(p) \lor \alpha^-(q)$.

As a result of these observations, we can claim closure of the notion of preciseness under the boolean operations.

Lemma 5 *If α is precise with respect to the assertions $p_1, \ldots, p_n$, then α is precise with respect to any boolean combination of these assertions.*

Proof: We have to show that if α is precise w.r.t.[2] p and q, then it is also precise w.r.t. $\neg p$ and $p \land q$. For the case of negation we have:

$$\alpha^+(\neg p) \quad \sim \quad \neg\alpha^-(p) \land map(V_A) \quad \sim \quad \neg\alpha^+(p) \land map(V_A) \quad \sim \quad \alpha^-(\neg p).$$

For the case of conjunction, preciseness is established by

$$\alpha^+(p \land q) \quad \sim \quad (\alpha^+(p) \land \alpha^+(q)) \quad \sim \quad (\alpha^-(p) \land \alpha^-(q)) \quad \sim \quad \alpha^-(p \land q).$$

▟

The notion of precision can be extended to temporal formulas, provided α is precise w.r.t all of their atomic sub-formulas.

[2] w.r.t. is an abbreviation for “with respect to”.

Lemma 6 *Let ψ be a temporal formula and α an abstraction mapping such that α is precise w.r.t. all the atomic sub-formulas of ψ, and α maps each concrete variable $x_\kappa \in X_\psi$ into the abstract variable $x_{\alpha_\tau(\kappa)}$. Then*

$$\begin{aligned} \alpha_\tau^+(\varphi) &\approx \alpha_\tau^-(\varphi) && (9)\\ \chi(\alpha_\tau^+(\varphi)) &\sim \alpha_\tau^+(\chi(\varphi)), && (10) \end{aligned}$$

for every φ, a sub-formula of ψ.

Proof: The formula $\alpha_\tau^+(\varphi)$ is obtained from φ by replacing the positive-polarity MSS's p within φ by $\alpha^+(p)$ and the negative-polarity MSS's q by $\alpha^-(q)$. In $\alpha_\tau^-(\varphi)$, all the positive-polarity MSS's p are replaced by $\alpha^-(p)$ and the negative-polarity MSS's q by $\alpha^+(q)$. Since α is precise w.r.t all the atomic formulas of φ, it follows by Lemma 5 that it is also precise w.r.t all the MSS's of φ. Therefore, $\alpha_\tau^+(\varphi)$ is congruent to $\alpha_\tau^-(\varphi)$. In such cases. we often write $\alpha_\tau(\varphi)$ to represent $\alpha_\tau^+(\varphi)$ (which is congruent to $\alpha_\tau^-(\varphi)$).

To establish Equation (10), we observe that every φ, a sub-formula of ψ is a boolean combination of atomic formulas and principally temporal formulas. Therefore, $\alpha_\tau(\varphi)$ can be obtained by replacing each atomic p by $\alpha(p)$ and each principally temporal κ by $\alpha_\tau(\kappa)$. Applying χ to $\alpha_\tau(\varphi)$, further replaces each $\alpha_\tau(\kappa)$ by $x_{\alpha_\tau(\kappa)}$. Therefore, the overall effect of computing $\chi(\alpha_\tau^+(\varphi))$ amounts to the replacement of each atomic p by $\alpha(p)$ and each principally temporal κ by $x_{\alpha_\tau(\kappa)}$. In comparison, the computation of $\alpha_\tau^+(\chi(\varphi))$ first performs the replacement of every κ by x_κ and only later replaces each atomic p by $\alpha(p)$ and each x_κ by $x_{\alpha_\tau(\kappa)}$. However, the overall effect of these two processes results in the same final formula. ▟

The notion of precision of the transformer α^+ can be generalized to the double abstraction α^{++}. We say that α is *doubly precise* w.r.t the assertion $p = p(V, V')$ if $\alpha^{++}(p) \sim \alpha^{--}(p)$, where

$$\alpha^{--}(p): \quad \forall V, V' \left(V_A = \mathcal{E}^\alpha(V) \land V_A' = \mathcal{E}^\alpha(V') \quad \rightarrow \quad p(V, V')\right) \land map(V_A) \land map(V_A').$$

The following lemma states some of the properties of this notion.

Lemma 7

1. *If α is precise with respect to $q = q(V)$, then it is doubly precise w.r.t q and q'.*

2. *If α is doubly precise with respect to the assertions $p_1, \ldots, p_n$, then α is doubly precise with respect to any boolean combination of these assertions.*

3. *If α is doubly precise w.r.t the assertions $p(V, V')$ and $q(V, V')$ and is (singly) precise w.r.t $r(V)$, then*

$$\begin{aligned} \alpha^{++}(p \land q) &\sim \alpha^{++}(p) \land \alpha^{++}(q) && (11)\\ \alpha^{++}(p \land r) &\sim \alpha^{++}(p) \land \alpha^{+}(r) && (12)\\ \alpha^{++}(p \land r') &\sim \alpha^{++}(p) \land \alpha^{+}(r)' && (13) \end{aligned}$$

It also follows from the definitions that if $p = p(V)$, then both $\alpha^{++}(p) \sim \alpha^{+}(p)$ and $\alpha^{++}(p') \sim \alpha^{+}(p)'$ hold without any precision assumptions about p.

We observe that if an implication is valid, we can apply the abstractions α^{+} and α^{++} to both sides of the implication.

Lemma 8 $$\models p \;\rightarrow\; q \quad \text{implies} \quad \left\{ \begin{array}{llll} \models & \alpha^{+}(p) & \rightarrow & \alpha^{+}(q) \qquad \text{and} \\ \models & \alpha^{++}(p) & \rightarrow & \alpha^{++}(q) \end{array} \right\}$$

Finally, we show that given an assertion $p(V)$, any abstraction mapping α can be augmented to be precise with respect to $p(V)$.

Claim 9 (Existence of precise abstractions) *Let α be a mapping from concrete states over V into abstract states over V_A. Let $V_A = U_A \cup \{B_p\}$, where B_p is a Boolean variable defined by $B_p = p(V)$. Then α is precise with respect to $p(V)$.*

Proof: The first direction $\alpha^{-}(p) \rightarrow \alpha^{+}(p)$ is valid for any assertion p and mapping α, with no precision requirement. To prove this direction, we first expand the definitions

$$\forall V : V_A = \mathcal{E}^{\alpha}(V) \rightarrow p(V) \wedge \exists V : V_A = \mathcal{E}^{\alpha}(V) \quad \rightarrow \quad \exists V : V_A = \mathcal{E}^{\alpha}(V) \wedge p(V).$$

Skolemizing $map(V_A)$ into $V_A = \mathcal{E}^{\alpha}(v)$ and instantiating the remaining quantifications into $V = v$, we get

$$(V_A = \mathcal{E}^{\alpha}(v) \rightarrow p(v)) \wedge (V_A = \mathcal{E}^{\alpha}(v)) \quad \rightarrow \quad (V_A = \mathcal{E}^{\alpha}(v) \wedge p(v))$$

which is obviously valid.

Next, we prove the second direction $\alpha^{+}(p) \rightarrow \alpha^{-}(p)$. Expanding the definitions, we get

$$\exists V : V_A = \mathcal{E}^{\alpha}(V) \wedge p(V) \quad \rightarrow \quad \forall V : V_A = \mathcal{E}^{\alpha}(V) \rightarrow p(V) \wedge \exists V : V_A = \mathcal{E}^{\alpha}(V)$$

Since $\exists V : V_A = \mathcal{E}^{\alpha}(V) \wedge p(V)$ implies $\exists V : V_A = \mathcal{E}^{\alpha}(V)$, we only have to show

$$\exists V : V_A = \mathcal{E}^{\alpha}(V) \wedge p(V) \quad \rightarrow \quad \forall V : V_A = \mathcal{E}^{\alpha}(V) \rightarrow p(V)$$

We split V_A into $U_A \cup \{B_p\}$, expanding $V_A = \mathcal{E}^{\alpha}(V)$ to $U_A = \mathcal{E}^{\alpha}_{U}(V) \wedge B_p = p$. Substituting the expansion, and Skolemizing both sides of the implication, we get

$$U_A = \mathcal{E}^{\alpha}_{U}(v_1) \wedge B_p = p(v_1) \wedge p(v_1) \quad \rightarrow \quad \Big[\big(U_A = \mathcal{E}^{\alpha}_{U}(v_2) \wedge B_p = p(v_2)\big) \rightarrow p(v_2)\Big]$$

which is equivalent to

$$U_A = \mathcal{E}^{\alpha}_{U}(v_1) \wedge B_p = p(v_1) \wedge p(v_1) \wedge U_A = \mathcal{E}^{\alpha}_{U}(v_2) \wedge B_p = p(v_2) \quad \rightarrow \quad p(v_2)$$

which is obviously valid, due to the chain of equalities $p(v_1) = \mathrm{T}$, $B_p = p(v_1)$ and $B_p = p(v_2)$.

■

7 Augmentation by Ranking and Progress Monitors

In the previous sections, we presented an example of successful finitary abstraction. However, there are cases when abstraction alone is inadequate for transforming an infinite-state system satisfying a property into a finite-state abstraction which maintains the property.

Before treating the general case, we will illustrate the problem and the proposed solution by two examples.

In Fig. 4, we present a simple looping program. The property we wish to verify is that program LOOP always terminates, independently of the initial value of the natural variable y.

```
            $y$: natural
$\ell_0$ :  while $y > 0$ do
              [ $\ell_1$ :  $y := y - 1$ ]
              [ $\ell_2$ :  skip         ]
$\ell_3$ :
```

Figure 4: Program LOOP.

A natural abstraction for the variable y is into the two-valued domain $\{zero, pos\}$. However, applying this abstraction yields the abstract program LOOP-ABS-1, presented in Fig. 5, where the abstract function *sub1* is defined by

$$\mathit{sub1}(Y) \quad = \quad \textbf{if } Y = \mathit{pos} \textbf{ then } \{\mathit{zero}, \mathit{pos}\} \textbf{ else } \mathit{zero}$$

```
            $Y$: $\{zero, pos\}$
$\ell_0$ :  while $Y = pos$ do
              [ $\ell_1$ :  $Y := sub1(Y)$ ]
              [ $\ell_2$ :  skip           ]
$\ell_3$ :
```

Figure 5: Program LOOP-ABS-1 abstracting program LOOP.

Unfortunately, program LOOP-ABS-1 need not terminate, because the function *sub1* can always choose to yield *pos* as a result.

To obtain a working abstraction, we first compose program LOOP with an additional module, to which we refer as the *ranking monitor* for variable y, as shown in Fig. 6.

The construct **always do** appearing in MONITOR means that the assignment which is the body of this construct is executed at *every* step. The comparison function $\mathit{diff}(y, y')$ is defined by

$$\mathit{diff}(y, y') \quad = \quad \mathit{sign}(y' - y) \quad = \quad \textbf{if } y < y' \textbf{ then } 1 \textbf{ else if } y = y' \textbf{ then } 0 \textbf{ else } -1.$$

```
                                   $y$: **natural**

  $\ell_0$: **while** $y > 0$ **do**                      $inc : \{-1,0,1\}$
        $\ell_1$: $y := y - 1$               |||    $m_0$: **always do**
        $\ell_2$: **skip**                                  $inc := \mathit{diff}(y, y')$
  $\ell_3$:

          — LOOP —                          — MONITOR $M_y$ —
```

Figure 6: Program LOOP composed with a ranking monitor.

Note that the expressions on the right-hand-side of the assignments in the monitor allow references to the *new* values of y as computed in the same step by the monitored program.

The presentation of the monitor module M_y in Fig. 6 is only for illustration purposes. The precise definition of this module is given by the following FDS:

$$M_y: \quad \left\langle \begin{array}{llll} V = & \{y : \textbf{natural};\ \mathit{inc} : \{-1,0,1\}\}, & \Theta : & \mathit{true}, \\ \rho : & \mathit{inc}' = \mathit{diff}(y, y'), & \mathcal{J} : & \emptyset, \quad \mathcal{C} : \ \{(\mathit{inc} < 0, \mathit{inc} > 0)\} \end{array} \right\rangle$$

Thus, at every step of the computation, module M_y compares the new value of y (y') with the current value, and sets variable *inc* to -1, 0, or 1, according to whether the value of y has decreased, stayed the same, or increased, respectively. This FDS has no justice requirements but has the single compassion requirement $(\mathit{inc} < 0, \mathit{inc} > 0)$ stating that y cannot decrease infinitely many times without also increasing infinitely many times. This requirement is a direct consequence of the fact that y ranges over the well-founded domain of the natural numbers, which does not allow an infinitely decreasing sequence.

It is possible to represent the composition of program LOOP with the ranking monitor M_y as (almost) equivalent to the sequential program presented in Fig. 7, where we have conjoined the repeated assignment of module M_y with every assignment of process LOOP. The "almost" qualification admits that we did not conjoin this assignment with the transition associated with location ℓ_0 which tests the value of y and decides when to terminate. In a fully formal treatment of this example, the assignment will also be conjoined to this testing transition.

```
                 $y$     : **natural**
                 $inc$   : $\{-1,0,1\}$

  $\ell_0$: **while** $y > 0$ **do**
        $\ell_1$:  $(y, inc)$  :=  $(y - 1,\ \mathit{diff}(y, y'))$
        $\ell_2$:       $inc$  :=          $\mathit{diff}(y, y')$
  $\ell_3$:
```

Figure 7: A sequential equivalent of the monitored program.

The abstraction of the program of Fig. 7 will abstract y into a variable Y ranging over $\{zero, pos\}$. The variable *inc*, ranging over the finite domain $\{-1, 0, 1\}$, is not abstracted. The resulting abstraction is presented in Fig. 8.

```
         $Y$            : $\{zero, pos\}$
         $inc$          : $\{-1, 0, 1\}$
         compassion     $(inc < 0, inc > 0)$
$\ell_0$ : while $Y = pos$ do
             [ $\ell_1$ : $(Y, inc) := (sub1(Y), -1)$
               $\ell_2$ :       $inc := 0$ ]
$\ell_3$ :
```

Figure 8: Abstracted Version of the monitored program – Program LOOP-ABS-2.

The explicit values of -1 and 0, assigned to variable *inc* in statements ℓ_1 and ℓ_2, respectively, are obtained automatically as part of the computation of the abstraction $\alpha^{++}(\rho)$.

Program LOOP-ABS-2 (Fig. 8) differs from program LOOP-ABS-1 (Fig. 5) by the additional compassion requirement $(inc < 0, inc > 0)$. However, it is this additional requirement which forces program LOOP-ABS-2 to terminate. This is because a run in which *sub1* always yields *pos* as a result is a run in which *inc* is negative infinitely many times (on every visit to ℓ_2) and is never positive beyond the first state. The fact that LOOP-ABS-2 always terminates can now be successfully model-checked.

7.1 More Complicated Cases

Next, we consider a more complicated case in which the ranking measuring the distance to termination is not a simple program variable but some function of the program variables.

In Fig. 9, we consider another always terminating program. To prove termination of this program we cannot take the value of y to be a never-increasing progress measure.

```
                 $y$: natural
$\ell_0$ : while $y > 1$ do
             [ $\ell_1$ : $y := y - 2$
               $\ell_2$ : $y := \{y + 1, y\}$
               $\ell_3$ : skip ]
$\ell_4$ :
```

Figure 9: Program SUB-ADD with a less trivial progress measure.

The assignment at statement ℓ_2 non-deterministically assigns to y the values $y + 1$ or y. Termination of such programs can always be established by identification of a *progress* *measure* that never increases and sometimes is guaranteed to decrease. For the simple case of program LOOP, y served as an adequate progress measure.

For program SUB-ADD, we must use a more complex progress measure. For example, we can use the progress measure $\delta : y + at\_\ell_2$ which never increases and always decreases on the execution of statement ℓ_1. Consequently, we can use the monitor presented in Fig. 10.

```
 define  $\delta = y + at\_\ell_2$
 $inc$      : $\{-1, 0, 1\}$
$m_0$ :  always do
             $inc := diff(\delta, \delta')$
```

Figure 10: A ranking monitor for program SUB-ADD.

Note that the only difference between ranking monitors is in the definition of the progress measure δ. The added compassion requirement is always the same, and is given by $(inc < 0, \quad inc > 0)$.

We can now abstract program SUB-ADD composed with its ranking monitor (Fig. 10), using the abstraction

$$Y \quad = \quad \textbf{if } y = 0 \textbf{ then } zero \textbf{ else if } y = 1 \textbf{ then } one \textbf{ else } large.$$

The resulting abstracted version is presented in Fig. 11, where the abstract functions *sub2* and *add1* are defined by

$$\begin{array}{lcl} sub2(Y) & = & \textbf{if } Y \in \{zero, one\} \textbf{ then } zero \textbf{ else } \{zero, one, large\} \\ add1(Y) & = & \textbf{if } Y = zero \textbf{ then } one \textbf{ else } large \end{array}$$

```
          $Y$                : $\{zero, one, large\}$
          $inc$              : $\{-1, 0, 1\}$
          compassion    $(inc < 0, inc > 0)$
$\ell_0$ :  while $Y = large$ do
          [ $\ell_1$ : $(Y, inc)$ := $(sub2(Y), -1)$
            $\ell_2$ : $(Y, inc)$ := $(\{add1(Y), Y\}, \{0, -1\})$
            $\ell_3$ :     $inc$ :=          $0$                    ]
$\ell_4$ :
```

Figure 11: The abstracted version of the monitored program SUB-ADD.

It is not difficult to see that model checking this program with the added compassion requirement will prove that the program always terminates.

The extension to the case that the progress measure ranges not over the naturals but over lexicographic tuples of naturals is straightforward.

7.2 The General Structure of a Ranking Monitor

Encouraged by these examples, we proceed to define the general structure of a ranking monitor and show that its augmentation to a verified system is safe, in the sense that all relevant temporal properties are preserved.

Let $(\mathcal{W}, \prec)$ be well-founded domain and δ be a ranking function, mapping the states of $\mathcal{D}$ into the well-founded domain.

A *ranking monitor* for a ranking function δ is an FDS M_δ of the following form:

$$M_\delta = \left\langle \begin{array}{llllll} V: & V_\mathcal{D}, \mathit{inc} : \{-1,0,1\}\}, & \Theta: & \mathit{true}, & & \\ \rho: & \mathit{inc}' = \mathit{diff}(\delta(V_\mathcal{D}), \delta(V'_\mathcal{D})), & \mathcal{J}: & \emptyset, & \mathcal{C}: & \{(\mathit{inc} < 0, \mathit{inc} > 0)\} \end{array} \right\rangle$$

7.3 When is it Safe to Augment?

There are cases in which even the more general ranking monitor is not sufficient, and we may have to augment the system by additional types of monitors. A most important requirement is that any such augmentation be safe.

Here, we identify general sufficient conditions which a monitor M should satisfy in order that its augmentation to a system $\mathcal{D}$ will be safe. Let M be an FDS with system variables V_M, and let $A \subseteq V_M$ be a subset of M's variables. We say that M is *accommodating for* $V_M - A$ ($\overline{A}$*-accommodating* for short) if, for every state sequence σ, there exists an A-variant of σ which is a computation of M. Thus, an accommodating FDS can, by merely reinterpreting the variables of A, transform any arbitrary state sequence σ into a computation of M. For example, the two ranking monitors we have considered above, i.e., M_y, and the monitor presented in Fig. 10, are both accommodating for $V_M - \{\mathit{inc}\}$.

An FDS M is said to be *accommodating for an* FDS $\mathcal{D}$ if M is accommodating for $V_M - A$, where $A \subseteq V_M$, and $V_\mathcal{D} \cap A = \emptyset$. The following claim states that if M is accommodating for $\mathcal{D}$, then the augmentation of $\mathcal{D}$ by M is safe, i.e., it preserves all the temporal properties of $\mathcal{D}$.

Claim 10 *Let M and $\mathcal{D}$ be two* FDS *'s such that M is accommodating for $\mathcal{D}$. Then, for every formula ψ relevant for $\mathcal{D}$, ψ is valid over $\mathcal{D}$ iff ψ is valid over $\mathcal{D} \,|||\, M$, the augmentation of $\mathcal{D}$ by M.*

Proof
In general, when we compare the set of computations of a system $\mathcal{D}$ with computations of a parallel composition of $\mathcal{D}$ with an arbitrary system M, we can only claim the one-side inclusion

$$\mathcal{C}\mathit{omp}(\mathcal{D} \,|||\, M)\Downarrow_\mathcal{D} \quad \subseteq \quad \mathcal{C}\mathit{omp}(\mathcal{D}).$$

That is, every computation of the composition $\mathcal{D} \,|||\, M$ is also a computation of $\mathcal{D}$ when projected onto the variables of $\mathcal{D}$. However, in the case of an accommodating monitor M which satisfies the premises of the claim, there is also an inclusion in the other direction. Namely, every computation of $\mathcal{D}$ can be extended to a computation of $\mathcal{D} \,|||\, M$. To see this, consider $\sigma : s_0, s_1, \ldots$, a computation of $\mathcal{D}$, in which we extended the states to assign arbitrary values to the variables in $V_M - V_{\mathcal{D}} \supseteq A$. Since, M is accommodating for $V_M - A$, we can reassign new values to the A-variables and obtain a new sequence $\widetilde{\sigma} : \widetilde{s}_0, \widetilde{s}_1, \ldots$, such that $\widetilde{\sigma}{\Downarrow}_M$ is a computation of M, while $\widetilde{\sigma}{\Downarrow}_{\mathcal{D}}$ is still a computation of $\mathcal{D}$. It follows that $\widetilde{s}$ is a computation of $\mathcal{D} \,|||\, M$.

We can therefore conclude that

$$\mathcal{C}omp(\mathcal{D} \,|||\, M){\Downarrow}_{\mathcal{D}} \quad = \quad \mathcal{C}omp(\mathcal{D}).$$

Since the validity of a $\mathcal{D}$-relevant ψ only depends on the interpretation given to the system variables of $\mathcal{D}$, the claim follows. ▟

We have argued above that a general ranking monitor M_δ is accommodating for $V_M - \{inc\}$. At the end of Section 4, we made a claim that can now be interpreted as saying that the tester T^{φ}_{true} is accommodating for $V_T - (X_\varphi \cup \{u\})$, where V_T represent the system variables of T^{φ}_{true}. We therefore conclude that augmentation of a system $\mathcal{D}$ with either a ranking monitor or a tester of the form T^{φ}_{true} is safe, i.e. preserves all temporal properties of $\mathcal{D}$.

In the most general case, we form a parallel combination of a tester of the form T^{φ}_{true} and a ranking monitor M_δ. We refer to such a composition $M = T^{\varphi}_{true} \,|||\, M_\delta$ as a *progress monitor*.

7.4 Verification by Augmented Finitary Abstraction

We can now formulate the method of *verification by augmented finitary abstraction* (VAA) as follows:

Verification by augmented finitary abstraction
To verify that ψ is $\mathcal{D}$-valid,

1. Optionally, choose a progress monitor FDS M which is accommodating for $\mathcal{D}$ and let $\mathcal{A} = \mathcal{D} \,|||\, M$. In case this step is skipped, we let $\mathcal{A} = \mathcal{D}$.

2. Choose a finitary state abstraction mapping α and calculate $\mathcal{A}^\alpha$ and ψ^α according to the recipes of Section 6.

3. Model check $\mathcal{A}^\alpha \models \psi^a$.

4. Infer $\mathcal{D} \models \psi$.

Corollary 11 (Soundness of the VAA method)
The VAA *method is sound.*

Proof
Assume that the VAA method has been applied successfully to system $\mathcal{D}$ and formula ψ. By Claim 4 and the success of step 3 we can conclude that $\mathcal{A} \models \psi$. By Claim 10 we obtain $\mathcal{D} \models \psi$. ▟

8 Completeness of the VAA Method

In the following sections, we prove the completeness of the VAA method. Let $\mathcal{D} = \langle V, \Theta, \rho, \mathcal{J}, \mathcal{C} \rangle$ be a (possibly infinite-state) FDS, and ψ be an LTL property such that $\mathcal{D} \models \psi$. Let α be a finitary abstraction mapping, and M be an FDS which is accommodating for $\mathcal{D}$. We say that (M, α) is an *adequate augmented abstraction* for $(\mathcal{D}, \psi)$, if $(\mathcal{D} \,|||\, M)^{\alpha} \models \psi^{\alpha}$. To establish the completeness of the VAA method we show that, for every FDS $\mathcal{D}$ and LTL-property ψ such that $\mathcal{D} \models \psi$, there exists an adequate augmented abstraction.

8.1 The Structure of the Completeness Proof

The proof proceeds along the following steps:

1. The verification problem: We are given a system $\mathcal{D}$ and a formula ψ, such that $\mathcal{D} \models \psi$.

2. Shifting fairness from the system to the property: We remove the fairness (both justice and compassion) requirements from the system and add them as an antecedent to the property ψ. Thus, we consider the modified FDS $\mathcal{D}^{-}$, obtained by emptying the justice and compassion sets, and the modified LTL formula $\Psi : \mathit{fair}(\mathcal{D}) \;\rightarrow\; \psi$, where

$$\mathit{fair}(\mathcal{D}) : \quad \bigwedge_{J \in \mathcal{J}} \Box \Diamond J \;\wedge \bigwedge_{(p,q) \in \mathcal{C}} (\Box \Diamond p \;\rightarrow\; \Box \Diamond q)$$

is the temporal encoding of the fairness requirements for the original FDS $\mathcal{D}$.

We then claim that $\mathcal{D} \models \psi$ iff $\mathcal{D}^{-} \models \Psi$ and proceed with the proof with Ψ and $\mathcal{D}^{-}$.

3. Constructing the tester $T_{\neg\Psi}$: Using the methods of Section 4, we construct the temporal tester $T_{\neg\Psi}$, which is a BDS characterizing all the sequences violating the formula Ψ.

Assume that the tester is given by $T_{\neg\Psi} = \langle V_T,\ \Theta_T,\ \rho_T,\ \{J_T\},\ \emptyset \rangle$, where (without loss of generality), $V_T = V_{\mathcal{D}} \cup X_{\Psi} \cup \{u\}$.

4. Compose system with tester: We form the synchronous parallel composition of $\mathcal{D}^{-}$ with $T_{\neg\Psi}$ to obtain a BDS $\mathcal{B}_{(\mathcal{D}^{-},\neg\Psi)} = \mathcal{D}^{-} \,|||\, T_{\neg\Psi}$ whose computations ($\mathcal{C}omp(\mathcal{B}_{(\mathcal{D}^{-},\neg\Psi)})$) are all the *counter-examples* to the $\mathcal{D}^{-}$-validity of Ψ, i.e., sequences violating Ψ which are also computations of $\mathcal{D}^{-}$. Since $\mathcal{D}^{-} \models \Psi$, the BDS $\mathcal{B}_{(\mathcal{D}^{-},\neg\Psi)}$ has no computations and is, therefore, infeasible ([VW94]).

5. Obtaining Φ and Δ: Based on claim 2, we identify an assertion and a ranking function, which satisfy the three premises of rule WELL (section 5) for the BDS $\mathcal{B}_{(\mathcal{D}^{-},\neg\Psi)}$. We denote these assertion and ranking function by Φ and Δ, respectively. Note that Φ and Δ may depend on the values of the variables in both $\mathcal{D}^{-}$ and $T_{\neg\Psi}$.

6. Constructing the progress monitor: Based on the tester $T_{\neg\Psi}$ and the ranking function Δ obtained in the previous steps (3 and 5), we define the progress monitor

$$M_{T,\Delta}: \quad T^{\Psi}_{true} \mid\mid\mid M_{\Delta}.$$

The monitor $M_{T,\Delta}$ is an FDS resulting from the synchronous parallel composition of $T^{\neg\Psi}_{true}$, the temporal tester for $\neg\Psi$ with its initial condition reset to $(u = 0)$, and the ranking monitor for M_{Δ}.

The progress monitor $M_{T,\Delta}$ is given by

$$M_{T,\Delta} \quad = \quad \left\langle \begin{array}{llll} V: & V_T \cup \{inc : \{-1, 0, 1\}\}, & & \\ \Theta: & u = 0, & \rho: & \rho_T \quad \wedge \quad inc' = diff(\Delta, \Delta')), \\ \mathcal{J}: & \{u = 0\} & \mathcal{C}: & \{(inc < 0, inc > 0)\} \end{array} \right\rangle \tag{14}$$

7. Augment, abstract, and conclude: As the magic abstraction mapping, we can take any finitary mapping α which is precise with respect to the assertion Φ obtained in step 5 and all the atomic sub-formulas appearing in Ψ. In addition, α should map each $x_\kappa \in X_{\neg\Psi}$ into $x_{\alpha_T(\kappa)}$, and should not abstract the variables u and inc, introduced in steps 3 and 6.

As prescribed by the general VAA method, we form the augmented system $\mathcal{D} \mid\mid\mid M_{T,\Delta}$ and compute the abstractions $(\mathcal{D} \mid\mid\mid M_{T,\Delta})^\alpha$ and ψ^α. We conclude the proof in section 9, by showing that $(\mathcal{D} \mid\mid\mid M_{T,\Delta})^\alpha \models \psi^\alpha$.

The diagram in Fig. 12 provides a graphical representation of the sequence of steps comprising the completeness proof.

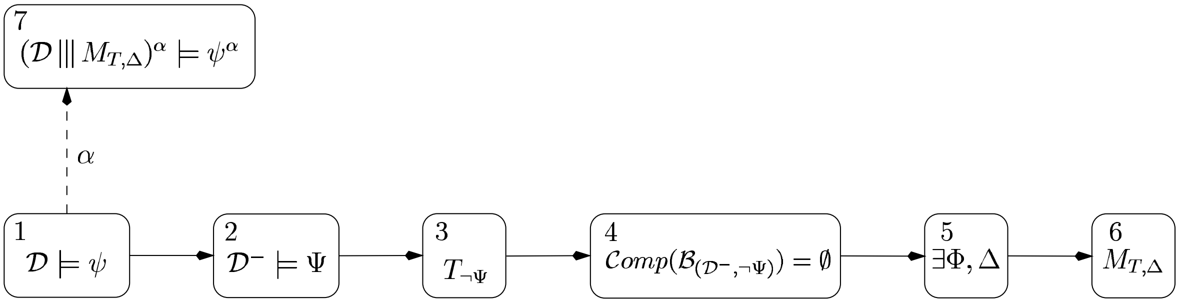
7
$(\mathcal{D} \mid\mid\mid M_{T,\Delta})^\alpha \models \psi^\alpha$
α
1
$\mathcal{D} \models \psi$
2
$\mathcal{D}^- \models \Psi$
3
$T_{\neg\Psi}$
4
$Comp(\mathcal{B}_{(\mathcal{D}^-,\neg\Psi)}) = \emptyset$
5
$\exists\Phi, \Delta$
6
$M_{T,\Delta}$

Figure 12: Scheme of the completeness proof.

8.2 A Characteristic Example

The whole construction will be illustrated on a single example. Consider program COND-TERM, presented in Fig. 13.
Statement ℓ_1 of this program nondeterministically assigns to variable x one of the values $-1, 1$. Program COND-TERM does not always terminate. In particular, it will not terminate if statement ℓ_1 always assigns to x the value 1. Consequently, the best we can claim for this program is the property of conditional termination which can be specified by

$$\psi: \quad \Diamond\Box(x < 0) \;\rightarrow\; \Diamond\, at\_\ell_3.$$

```
          $y$: natural
          $x$: $\{-1, 1\}$
$\ell_0$: while $y > 0$ do
            [ $\ell_1$: $x := \pm 1$
              $\ell_2$: $y := y + x$ ]
$\ell_3$:
```

Figure 13: Program COND-TERM.

This property states that if, from a certain point on, x remains negative, then the program will terminate. It is not difficult to see that this property is valid for program COND-TERM.

Since program COND-TERM is a sequential program, it is associated with no fairness requirement. Therefore, step 2 which shifts the fairness requirements from the system to the property is vacuous, and we have that $\mathcal{D}^- = \mathcal{D}$ and $\Psi = \psi$.

Step 3 of the proof scheme constructs a temporal tester $T_{\neg\Psi}$, which characterizes all the sequences violating ψ.

Following the construction described in Section 4, we obtain the BDS $T_{\neg\psi}$, given by:

$$
\begin{array}{ll}
V: & \pi, x : \textbf{natural}; \quad f_1, g_2, f_3 : \textbf{boolean}; \quad u : [0..3] \\
\Theta_{\neg\psi}: & u = 0 \ \wedge \ f_1 \ \wedge \ \neg f_3 \\
\rho_{\neg\psi}: & \left(\begin{array}{lllllll}
f_1 & \leftrightarrow & g_2 & \vee & f_1' & & \wedge \\
g_2 & \leftrightarrow & x < 0 & \wedge & g_2' & & \wedge \\
f_3 & \leftrightarrow & at\_\ell_3 & \vee & f_3' & & \wedge \\
u' = & \multicolumn{6}{l}{\left[\begin{array}{ll}
\textbf{case} & \\
\quad u = 0 & : 1\,; \\
\quad u = 1 \ \wedge \ (g_2 \ \vee \ \neg f_1) & : 2\,; \\
\quad u = 2 \ \wedge \ (x \geq 0 \ \vee \ g_2) & : 3\,; \\
\quad u = 3 \ \wedge \ (at\_\ell_3 \ \vee \ \neg f_3) & : 0\,; \\
\quad true & : u\,; \\
\textbf{esac} &
\end{array}\right]}
\end{array}\right) \\
J: & u = 0
\end{array}
$$

Step 4 of the construction forms the parallel composition of $\mathcal{D} = \mathcal{D}^-$ and $T_{\neg\Psi}$ to obtain the combined BDS $\mathcal{B}_{(\mathcal{D},\neg\Psi)} = \mathcal{D} \,|||\, T_{\neg\Psi}$. We claim that the system $\mathcal{B}_{(\mathcal{D},\neg\Psi)}$ has no computations. Assume to the contrary, that σ is a computation of $\mathcal{B}_{(\mathcal{D},\neg\Psi)}$. To be a computation, σ must contain infinitely many states in which $u = 0$. According to the initial condition, f_1 is initially true, while f_3 is initially false. By the transition relation for f_1 and the condition for getting out of $u = 1$, there must exist a position $j \geq 0$ such that $g_2 = 1$ at j. By the transition relation for g_2, it follows that $x < 0$ for all positions $k \geq j$. This means that, from j on, all executions of statement ℓ_2 cause y to decrease. Since a natural number cannot decrease infinitely many times, the while loop of the program must terminate, and the execution must reach location ℓ_3, which by $f_3 = 0$, is impossible.

According to step 5, we should be able to identify an assertion Φ which is an invariant of $\mathcal{B}_{(\mathcal{D}^-,\neg\Psi)}$, and a progress measure Δ. Indeed, for our example, an appropriate invariant assertion is

$$\Phi : (f_1 \vee\ g_2)\ \wedge\ \neg f_3\ \wedge\ (u > 1 \rightarrow\ g_2)\ \wedge\ (\pi \in \{1,2\} \rightarrow\ y > 0)$$

while a progress measure can be given by

$$\Delta : \left[\begin{array}{lr} \textbf{case} & \\ \quad g_2 : (0,\ 3y + 2at\_\ell_0 + at\_\ell_1) & ; \\ \quad 1\ \ : \hfill (1,\ 0) & ; \\ \textbf{esac} & \end{array}\right]$$

It is not difficult to see that any transition taken from a Φ-state is guaranteed not to increase Δ. If such a transition leads to a state in which $u = 0$ then Δ must decrease.

In step 6, we use the tester $T^{\neg\Psi}_{true}$, and the progress measure Δ to construct the progress monitor $M_{T,\Delta}$ given by

$$M_{T,\Delta} : \left\langle \begin{array}{rlrl} V_M : & \multicolumn{3}{l}{\{\pi, x : \textbf{natural},\ f_1, g_2, f_3 : \textbf{boolean},\ u : [0..3],\ inc : \{-1,0,1\}\,\}} \\ \Theta_M : & u = 0 & \rho_M : & \rho_{\neg\psi}\ \wedge\ inc' = diff(\Delta, \Delta') \\ \mathcal{J} : & u = 0 & \mathcal{C} : & \{(inc < 0, inc > 0)\} \end{array} \right\rangle$$

Next, we form the composition $\mathcal{D} \,|||\, M_{T,\Delta}$, and then compute the abstraction mapping α. To obtain a finitary mapping, we introduce a fresh Boolean variable $B_{y>0}$ with the definition $B_{y>0} = (y > 0)$. Applying the abstraction α to $\mathcal{D} \,|||\, M_{T,\Delta}$, we obtain an abstracted finite-state system equivalent to the program presented in Fig. 14.

$$\begin{array}{ll} B_{y>0}\!: & \textbf{boolean} \\ x\!: & \{-1,1\} \end{array}$$

$$\left[\begin{array}{l} \ell_0 :\ \ \textbf{while}\ B_{y>0}\ \textbf{do} \\ \quad \left[\begin{array}{l} \ell_1 :\ \ x := \pm 1 \\ \ell_2 :\ \ B_{y>0} := \left[\begin{array}{lr} \textbf{case} & \\ \quad x > 0\ \ : & 1\ ; \\ \quad \neg B_{y>0} : & 0\ ; \\ \quad 1 \quad\ \ : & \{0,1\}\ ; \\ \textbf{esac} & \end{array}\right] \end{array}\right] \\ \ell_3 : \end{array}\right] \ \ |||\ \ \left[\begin{array}{ll} F_1, G_2, F_3 : & \textbf{boolean} \\ u : & [0..3]\ \textbf{where}\ u = 0 \\ inc : & \{-1,0,1\} \\ \textbf{justice} & \quad u = 0 \\ \multicolumn{2}{l}{\textbf{compassion}\ (inc < 0,\ inc > 0)} \\ \multicolumn{2}{l}{\textbf{always do}} \\ \multicolumn{2}{l}{\left[\begin{array}{l} \alpha^{++}(\rho_{\neg\psi})\ \ \wedge \\ inc' = \left[\begin{array}{lr} \textbf{case} & \\ \quad \neg G_2\ \wedge\ G_2' & : -1\ ; \\ \quad \neg G_2\ \vee\ at\_\ell_3 & :\ \ 0\ ; \\ \quad at\_\ell_{0,1} & : -1\ ; \\ \quad x < 0\ \wedge\ B_{y>0} & : -1\ ; \\ \quad 1 & :\ \ 1\ ; \\ \textbf{esac} & \end{array}\right] \end{array}\right]} \end{array}\right]$$

Figure 14: Program ABS-COND-TERM, the augmented abstracted version of program COND-TERM.

The variables F_1, G_2, F_3 are the abstract versions of f_1, g_2, and f_3, respectively. Note that, like $\mathcal{D} \,|||\, M_{T,\Delta}$, system ABS-COND-TERM is a parallel composition of 3 components: the abstraction of program COND-TERM, the abstraction of the tester T^{Ψ}_{true}, and the abstraction of the monitor, taking into account its joint behavior with the other two components.

Clearly, system ABS-COND-TERM is a finite-state system and satisfies the property

$$\psi: \quad \diamondsuit \, \square (x < 0) \;\rightarrow\; \diamondsuit \, \mathit{at\_}\ell_3.$$

To see that ABS-COND-TERM satisfies the property ψ, assume, to the contrary, that there exists a computation σ of ABS-COND-TERM which satisfies $\diamondsuit \, \square (x < 0)$ but never reaches location ℓ_3. In this case, the initial values of f_1 and f_3 must be 1 and 0, respectively. The justice requirement with respect to u cannot be satisfied in such a case, unless g_2 eventually assume the value 1. Once this happens, *inc* is constantly -1 from this point on. This violates the compassion requirement with respect to *inc*. It follows that σ cannot be a computation.

9 The Abstracted System Satisfies the Abstracted Property

In the following we prove the completeness of the VAA method.

9.1 The Completeness Statement

Following is the completeness claim:

Claim 12 (Completeness of VAA) *Let $\mathcal{D} : \langle V_{\mathcal{D}}, \Theta_{\mathcal{D}}, \rho_{\mathcal{D}}, \mathcal{J}_{\mathcal{D}}, \mathcal{C}_{\mathcal{D}} \rangle$ be an* FDS *and ψ be a temporal formula such that $\mathcal{D} \models \psi$. Then, there exists an adequate augmented abstraction (M, α) such that $(\mathcal{D} || M)^{\alpha} \models \psi^{\alpha}$*

As progress monitor of our adequate augmented abstraction, we take $M_{T,\Delta}$, as defined in Equation (14). Recall that $M_{T,\Delta} = T^{\Psi}_{\mathit{true}} ||| M_{\Delta}$. As argued in Subsection 7.3, the tester T^{Ψ}_{true} is accommodating for $\mathcal{D}$ and M_{Δ} is accommodating for $\mathcal{D} ||| T^{\Psi}_{\mathit{true}}$. It follows that $M_{T,\Delta} = T^{\Psi}_{\mathit{true}} ||| M_{\Delta}$ is accommodating for $\mathcal{D}$.

Let us denote by $\mathcal{A}$ the augmented system $\mathcal{D} ||| M_{T,\Delta} = \mathcal{D} ||| M_{\Delta} ||| T^{\Psi}_{\mathit{true}}$. The components of this system are given by

$$\begin{array}{rl}
V_{\mathcal{A}}: & V_{\mathcal{D}} \cup X_{\Psi} \cup \{u, \mathit{inc}\} \\
\Theta_{\mathcal{A}}: & \Theta_{\mathcal{D}} \;\wedge\; u = 0 \\
\rho_{\mathcal{A}}: & \rho_{\mathcal{D}} \;\wedge\; \rho_{T} \;\wedge\; \rho_{\Delta} \\
\mathcal{J}_{\mathcal{A}}: & \mathcal{J}_{\mathcal{D}} \cup \{u = 0\} \\
\mathcal{C}_{\mathcal{A}}: & \mathcal{C}_{\mathcal{D}} \cup \{(\mathit{inc} < 0, \mathit{inc} > 0)\},
\end{array}$$

where ρ_{T} is the transition relation of T^{Ψ}_{true}, which is equal to the transition relation of $T_{\neg\Psi}$, and $\rho_{\Delta} : \mathit{inc}' = \mathit{diff}(\Delta, \Delta')$.

Let α be a finitary abstraction which is precise with respect to Φ and all the atomic subformulas of Ψ, maps each $x_{\kappa} \in X_{\neg\Psi}$ into $x_{\alpha_{\tau}(\kappa)}$, and does not abstract any of the auxiliary variables $\{u, inc\}$. In the following, we show that $\mathcal{A}^\alpha \models \psi^\alpha$, that is, the abstracted formula ψ^α is valid over all computations of the abstracted augmented system $\mathcal{A}^\alpha$.

Note that the requirement that α maps each $x_\kappa \in X_{\neg\Psi}$ into $x_{\alpha_T(\kappa)}$, implies that α is precise with respect to the Boolean variables in $X_{\neg\Psi}$, when viewed as propositional formulas.

9.2 Abstracting the Premises of Rule WELL

The proof is based on the abstraction of premises W1–W3 of rule WELL , applied to the BDS $\mathcal{B}_{(\mathcal{D}^-, \neg\Psi)}$ (Section 5).

Let us reconsider premises W1–W3 of rule WELL, which are known to be valid for our choice of Φ and Δ. The initial condition $\Theta_\mathcal{B}$ and transition relation $\rho_\mathcal{B}$ of $\mathcal{B}_{(\mathcal{D}^-, \neg\Psi)}$, are given by

$$\begin{array}{ll} \Theta_\mathcal{B}: & \Theta_\mathcal{D} \wedge u = 0 \wedge \neg\chi(\Psi) \\ \rho_\mathcal{B}: & \rho_\mathcal{D} \wedge \rho_T \end{array}$$

Recall that, since $\Psi = \Big(\bigwedge\limits_{J \in \mathcal{J}} \Box\Diamond J \wedge \bigwedge\limits_{(p,q)\in\mathcal{C}} (\Box\Diamond p \rightarrow \Box\Diamond q)\Big) \rightarrow \psi$, it follows that

$$\neg\chi(\Psi) \quad = \quad \underbrace{\bigwedge_{J\in\mathcal{J}} x_{\Box\Diamond J} \wedge \bigwedge_{(p,q)\in\mathcal{C}} (x_{\Box\Diamond p} \rightarrow x_{\Box\Diamond q})}_{\chi(fair(\mathcal{D}))} \wedge \neg\chi(\psi).$$

From the implication

$$inc' = diff(\Delta, \Delta') \quad \rightarrow \quad \left[\begin{array}{lll} \Delta' \preceq \Delta & \rightarrow & inc' \leq 0 \ \wedge \\ \Delta' \prec \Delta & \rightarrow & inc' < 0 \end{array}\right]$$

and premises $W1 - W3$ applied to $\mathcal{B}_{(\mathcal{D}^-, \neg\Psi)}$, we can obtain the following three valid implications:

$$\begin{array}{lll} \text{U1.} & \Theta_\mathcal{A} \wedge \neg\chi(\Psi) & \rightarrow \ \Phi \\ \text{U2.} & \rho_\mathcal{A} \wedge \Phi & \rightarrow \ \Phi' \wedge inc' \leq 0 \\ \text{U3.} & \rho_\mathcal{A} \wedge \Phi \wedge (u' = 0) & \rightarrow \ \Phi' \wedge inc' < 0. \end{array}$$

Based on lemma 8, we can apply α^+ to both sides of U1 and apply α^{++} to both sides of U2 and U3. We then simplify the right-hand sides, using the fact that $\alpha^{++}(p') \sim \alpha^+(p)'$, and that α does not abstract inc. Next, we use the fact that α is precise w.r.t. Φ, the variables in X_Ψ, and the atomic sub-formulas of Ψ, and that α does not abstract the variables in $\{u, inc\}$, in order to distribute the abstraction over the conjunctions on the left-hand sides of the implications, based on Equation (8) and Lemma 7. These transformations and simplifications lead to the following three valid abstract implications:

$$\begin{array}{lll} \text{V1.} & \alpha^+(\Theta_\mathcal{A}) \wedge \alpha(\chi(fair(\mathcal{D}))) \wedge \neg\chi(\psi^\alpha) & \rightarrow \ \alpha(\Phi) \\ \text{V2.} & \alpha^{++}(\rho_\mathcal{A}) \wedge \alpha(\Phi) & \rightarrow \ \alpha(\Phi)' \wedge inc' \leq 0 \\ \text{V3.} & \alpha^{++}(\rho_\mathcal{A}) \wedge \alpha(\Phi) \wedge (u' = 0) & \rightarrow \ \alpha(\Phi)' \wedge inc' < 0. \end{array}$$

The simplification of the abstraction $(\neg\chi(\Psi))^\alpha$ into $\alpha(\chi(fair(\mathcal{D}))) \wedge \neg\chi(\psi^\alpha)$ can be done in two steps. In the first step we rewrite $\neg\chi(\Psi)$ as $\chi(fair(\mathcal{D})) \wedge \neg\chi(\psi)$. Then we simplify the abstraction, based on Equation (8) and the fact that α is precise with respect to the atomic formulas within Ψ and the Boolean variables in X_Ψ.

9.3 No Computation of $\mathcal{A}^\alpha$ can Violate ψ^α

We will show that no computation of $\mathcal{A}^\alpha$ can violate ψ^α. Assume, to the contrary, that there exists an $\mathcal{A}^\alpha$-computation σ which violates ψ^α.

The proof proceeds in several steps.

9.3.1 The sequence σ is a computation of $(T^{\Psi}_{true})^\alpha$

We will now show that the sequence σ_{A}, assumed to be a computation of $\mathcal{A}^\alpha$ is also a computation of $(T^{\Psi}_{true})^\alpha$.

Computing, we find out that $\mathcal{A}^\alpha$ and $(T^{\Psi}_{true})^\alpha$ are given by

$$\begin{array}{rllllll} \mathcal{A}^\alpha: & \langle & V^A_{\mathcal{A}}, & \alpha^+(\Theta_{\mathcal{D}}) \ \wedge \ u=0, & \alpha^{++}(\rho_{\mathcal{D}} \ \wedge \ \rho_{\Delta} \ \wedge \ \rho_{T}), & \mathcal{J}^\alpha_{\mathcal{A}}, & \mathcal{C}^\alpha_{\mathcal{A}} \ \rangle \\ (T^{\Psi}_{true})^\alpha: & \langle & V^A_{\mathcal{A}}, & u=0, & \alpha^{++}(\rho_{T}), & u=0, & \emptyset \ \rangle, \end{array}$$

where $V^A_{\mathcal{A}}$ is the set of abstract variables for $\mathcal{A}^\alpha$ and $\mathcal{J}^\alpha_{\mathcal{A}}$, $\mathcal{C}^\alpha_{\mathcal{A}}$ are the abstracted versions of the fairness sets. Without loss of generality, we extended the set of systems variables of $(T^{\Psi}_{true})^\alpha$ to include all of $V^A_{\mathcal{A}}$.

Since $\alpha^+(\Theta_{\mathcal{D}}) \ \wedge \ u=0$ implies $u=0$, $\alpha^{++}(\rho_{\mathcal{D}} \ \wedge \ \rho_{\Delta} \ \wedge \ \rho_{T})$ implies $\alpha^{++}(\rho_{T})$, and $u=0$ is one of the justice requirements included in $\mathcal{J}^\alpha_{\mathcal{A}}$, it follows that every computation of $\mathcal{A}^\alpha$ is also a computation of $(T^{\Psi}_{true})^\alpha$.

9.3.2 The sequence σ is also a computation of $T^{\Psi^\alpha}_{true}$

In the preceding discussion, we showed that σ is a computation of $(T^{\Psi}_{true})^\alpha$. Here we show that, in fact, it is also a computation of $T^{\Psi^\alpha}_{true}$. Note that the difference between the two systems is that, while forming $(T^{\Psi}_{true})^\alpha$, we first constructed the tester for Ψ and then abstracted the resulting system. To generate $T^{\Psi^\alpha}_{true}$, we first compute $\Psi^\alpha = \alpha^-_{T}(\Psi)$ the abstracted temporal formula, following the recipe of Subsection 6.1, and only then construct a tester for the formula Ψ^α, following the recipe prescribed in Section 4.

The claim follows from the stronger statement

$$(T^{\Psi}_{true})^\alpha \quad \sim \quad T^{\Psi^\alpha}_{true},$$

which states that the two systems are actually equivalent, despite the different orders in which we applied the processes of abstraction and tester construction.

Obviously, $(T^{\Psi}_{true})^\alpha$ and $T^{\Psi^\alpha}_{true}$ agree on the set of their variables, their initial condition which is $u=0$, the justice set which consists of the single requirement $u=0$, and the compassion set which is empty for both.

It only remains to compare the transition relations of the two systems, which we denote by ρ_{after} for $(T^{\Psi}_{true})^\alpha$ and ρ_{before} for $T^{\Psi^\alpha}_{true}$. Recall that the transition relation for T^{Ψ}_{true} is given by a conjunction of clauses, containing one clause C_κ for each principally temporal sub-formula $\kappa \in \Psi$ and one big clause C_u for the variable u. It can be shown that the transition relation for $(T^{\Psi}_{true})^\alpha$ is given by

$$\rho_{after} = \alpha^{++}(C_u \ \wedge \ \bigwedge_{\kappa \in \Psi} C_\kappa).$$

Since C_u and each of the C_κ are formed as a boolean combination of $X_\Psi \cup \{u\}$ and their primed versions, and atomic formulas which appear in Ψ, the mapping α is doubly precise w.r.t C_u and all C_κ's. We can use Lemma 7 to rewrite ρ_{after} as

$$\rho_{\mathit{after}} = \alpha^{++}(C_u) \;\wedge\; \bigwedge_{\kappa \in \Psi} \alpha^{++}(C_\kappa).$$

In comparison, the transition relation for $T^{\Psi^\alpha}_{\mathit{true}}$ is given by a similar conjunction

$$\rho_{\mathit{before}} = \widetilde{C}_u \;\wedge\; \bigwedge_{\kappa \in \Psi} \widetilde{C}_\kappa,$$

where, due to precision, $\alpha^{++}(C_\kappa)$ is equivalent to $\widetilde{C}_\kappa$ for every $\kappa \in \Psi$, and $\alpha^{++}(C_u)$ is equivalent to $\widetilde{C}_u$.

To illustrate this point, consider the case that $\kappa = p\mathcal{U}q$. For this case, $\alpha^{++}(C_\kappa)$ is given by $x_{\alpha_{\mathcal{T}}(p\mathcal{U}q)} = \alpha(\chi(q)) \;\vee\; (\alpha(\chi(p)) \;\wedge\; x'_{\alpha_{\mathcal{T}}(p\mathcal{U}q)})$ while $\widetilde{C}_\kappa$ is given by $x_{\alpha_{\mathcal{T}}(p\mathcal{U}q)} = \chi(\alpha(q)) \;\vee\; (\chi(\alpha(p)) \;\wedge\; x'_{\alpha_{\mathcal{T}}(p\mathcal{U}q)})$. Since α is precise with respect to all atomic subformulas of Ψ and their boolean combinations, in particular w.r.t. p and q, it is clear (see Equation (10)) that $\widetilde{C}_\kappa$ is equivalent to $\alpha^{++}(C_\kappa)$.

We conclude that $\rho_{\mathit{after}} \sim \rho_{\mathit{before}}$ and, therefore, $(T^{\Psi}_{\mathit{true}})^\alpha$ is equivalent to $T^{\Psi^\alpha}_{\mathit{true}}$.

9.3.3 The sequence σ is a computation of $T_{\neg\Psi^\alpha}$

Since σ is a computation of $T^{\Psi^\alpha}_{\mathit{true}}$, it must be either a computation of T_{Ψ^α} or a computation of $T_{\neg\Psi^\alpha}$, depending on the initial value of $\chi(\Psi^\alpha)$.

Assume for the moment that σ is a computation of T_{Ψ^α}. Then σ must satisfy the formula $\Psi^\alpha = \mathit{fair}(\mathcal{D})^\alpha \;\rightarrow\; \psi^\alpha$, where

$$\mathit{fair}(\mathcal{D})^\alpha = \bigwedge_{J \in \mathcal{J}_\mathcal{D}} \Box\Diamond(\alpha(J)) \;\wedge\; \bigwedge_{(p,q) \in \mathcal{C}_\mathcal{D}} (\Box\Diamond(\alpha(q) \;\rightarrow\; \Box\Diamond(\alpha(q))).$$

Note that since α is precise with respect to all J's, p's and q's (being precise w.r.t. all the state sub-formulas of Ψ) we do not have to distinguish between α^+ and α^-. As σ is also a computation of $(\mathcal{D} \,|||\, M_\Delta)^\alpha$, it must satisfy the fairness requirement $\mathit{fair}(\mathcal{D})^\alpha$, leading to the fact that σ satisfies ψ^α in contradiction to our initial contrary assumption that σ violates ψ^α.

We therefore conclude that $\sigma : s_0, s_1, \ldots,$ is a computation of $T_{\neg\Psi^\alpha}$. In particular, s_0 satisfies $\alpha(\chi(\mathit{fair}(\mathcal{D}))) \;\wedge\; \neg\chi(\psi^\alpha)$.

9.3.4 The sequence σ cannot be a computation of $\mathcal{A}^\alpha$

We proceed to show that σ cannot be a computation of $\mathcal{A}^\alpha$. We use the implications V1–V3 to show that the assertion $\alpha(\Phi)$ is an invariant of σ.

Since we established that the first state of σ satisfies $\alpha(\chi(\mathit{fair}(\mathcal{D}))) \;\wedge\; \neg\chi(\psi^\alpha)$ and, being a computations of $\mathcal{A}^\alpha$ it certainly satisfies $\alpha^+(\Theta_\mathcal{A})$, we conclude by V1 that the first state of σ satisfies $\alpha(\Phi)$. Proceeding from each state s_j of σ to its successor s_{j+1}, which must be an $\alpha^{++}(\rho_\mathcal{A})$-successor of s_j, we see that $\alpha(\Phi)$ keeps propagating. It follows that $\alpha(\Phi)$ is an invariant of σ, i.e, every state s_i is σ satisfies $\alpha(\Phi)$.

Since σ is a computation of $T_{\neg\Psi^\alpha}$, it must contain infinitely many states which satisfy $\alpha(J_T) : u = 0$. According to implications V2 and V3, the variable *inc* is never positive, and is negative infinitely many times. Such a behavior contradicts the compassion requirement $(inc < 0, inc > 0)$ associated with $\mathcal{A}^\alpha$. Thus, σ cannot be a computation of $\mathcal{A}^\alpha$.

We conclude that all computations of $\mathcal{A}^\alpha$ must satisfy ψ^α.

10 Conclusions

We have presented a method for verification by augmented finitary abstraction by which, in order to verify that a (potentially infinite-state) system satisfies a temporal property, one first augments the system with a non-constraining progress monitor and then abstracts the augmented system and the temporal specification into a finite-state verification problem, which can be resolved by model checking. The method has been shown to be sound and complete.

In principle, the established completeness promotes the VAA method to the status of a viable alternative to the verification of infinite-state reactive systems by temporal deduction. Some potential users of formal verification may find the activity of devising good abstraction mappings more tractable (and similar to programming) than the design of auxiliary invariants. However, on a deeper level it is possible to argue that this is only a formal shift and that the same amount of ingenuity and deep understanding of the analyzed system is still required for effective verification as in the practice of temporal deduction methods.

The development of the VAA theory calls for additional research in the implementation of these methods. In particular, there is a strong need for devising heuristics for the automatic generation of effective abstraction mappings and corresponding augmenting monitors.

Acknowledgment: We gratefully acknowledge the many useful discussions and insightful observations by Moshe Vardi which helped us clarify the main issues considered in this paper. We thank Saddek Bensalem for his helpful comments and for running many of our examples on the automatic abstraction system INVEST [BLO98b]. We also thank the referees for their insightful comments.

References

[BBM95] N. Bjørner, I.A. Browne, and Z. Manna. Automatic generation of invariants and intermediate assertions. In 1st *Intl. Conf. on Principles and Practice of Constraint Programming*, volume 976 of *Lect. Notes in Comp. Sci.*, pages 589–623. Springer-Verlag, 1995.

[BLO98a] S. Bensalem, Y. Lakhnech, and S. Owre. Abstractions of infinite state systems compositionally and automatically. In A.J. Hu and M.Y. Vardi, editors, *Proc. 10th Intl. Conference on Computer Aided Verification (CAV'98)*, volume 1427 of *Lect. Notes in Comp. Sci.*, pages 319–331. Springer-Verlag, 1998.

[BLO98b] S. Bensalem, Y. Lakhnech, and S. Owre. A tool for the verification of invariants. In A.J. Hu and M.Y. Vardi, editors, *Proc. 10th Intl. Conference on*

Computer Aided Verification (CAV'98), volume 1427 of *Lect. Notes in Comp. Sci.*, pages 505–510. Springer-Verlag, 1998.

[BMS95] I.A. Browne, Z. Manna, and H.B. Sipma. Generalized verification diagrams. In P.S. Thiagarajan, editor, *15th Conference on the Foundations of Software Technology and Theoretical Computer Science*, volume 1026 of *Lect. Notes in Comp. Sci.*, pages 484–498. Springer-Verlag, 1995.

[CC77] P. Cousot and R. Cousot. Abstract interpretation: A unified lattice model for static analysis of programs by construction or approximation of fixpoints. In *Proceedings of the 4th Annual Symposium on Principles of Programming Languages*. ACM Press, 1977.

[CGH94] E.M. Clarke, O. Grumberg, and K. Hamaguchi. Another look at LTL model checking. In D. L. Dill, editor, *Proc. 6th Conference on Computer Aided Verification*, volume 818 of *Lect. Notes in Comp. Sci.*, pages 415–427. Springer-Verlag, 1994.

[CGL94] E.M. Clarke, O. Grumberg, and D.E. Long. Model checking and abstraction. *ACM Trans. Prog. Lang. Sys.*, 16(5):1512–1542, 1994.

[CGL96] E.M. Clarke, O. Grumberg, and D.E. Long. Model checking. In *Model Checking, Abstraction and Composition*, volume 152 of *Nato ASI Series F*, pages 477–498. Springer-Verlag, 1996.

[CH78] P. Cousot and N. Halbwachs. Automatic discovery of linear restraints among variables of a program. In *Proc. 5th ACM Symp. Princ. of Prog. Lang.*, pages 84–96, 1978.

[Cho74] Y. Choueka. Theories of automata on ω-tapes: A simplified approach. *J. Comp. Systems Sci.*, 8:117–141, 1974.

[DGG97] D. Dams, R. Gerth, and O. Grumberg. Abstract interpretation of reactive systems. *ACM Trans. Prog. Lang. Sys.*, 19(2), 1997.

[KP98a] Y. Kesten and A. Pnueli. Deductive verification of fair discrete systems. Technical report, Minerva Center for the Verification of Reactive Systems at the Weizmann Institute, 1998.

[KP98b] Y. Kesten and A. Pnueli. Modularization and abstraction: The keys to formal verification. In L. Brim, J. Gruska, and J. Zlatuska, editors, *The 23rd International Symposium on Mathematical Foundations of Computer Science*, volume 1450 of *Lect. Notes in Comp. Sci.*, pages 54–71. Springer-Verlag, 1998.

[KPR98] Y. Kesten, A. Pnueli, and L. Raviv. Algorithmic verification of linear temporal logic specifications. In K.G. Larsen, S. Skyum, and G. Winskel, editors, *Proc. 25th Int. Colloq. Aut. Lang. Prog.*, volume 1443 of *Lect. Notes in Comp. Sci.*, pages 1–16. Springer-Verlag, 1998.

[LGS$^+$95] C. Loiseaux, S. Graf, J. Sifakis, A. Bouajjani, and S. Bensalem. Property preserving abstractions for the verification of concurrent systems. *Formal Methods in System Design*, 6(1):11–44, 1995.

[LP85] O. Lichtenstein and A. Pnueli. Checking that finite-state concurrent programs satisfy their linear specification. In *Proc. 12th ACM Symp. Princ. of Prog. Lang.*, pages 97–107, 1985.

[LPS81] D. Lehmann, A. Pnueli, and J. Stavi. Impartiality, justice and fairness: The ethics of concurrent termination. In *Proc. 8th Int. Colloq. Aut. Lang. Prog.*, volume 115 of *Lect. Notes in Comp. Sci.*, pages 264–277. Springer-Verlag, 1981.

[MAB$^+$94] Z. Manna, A. Anuchitanukul, N. Bjørner, A. Browne, E. Chang, M. Colón, L. De Alfaro, H. Devarajan, H. Sipma, and T.E. Uribe. STeP: The Stanford Temporal Prover. Technical Report STAN-CS-TR-94-1518, Dept. of Comp. Sci., Stanford University, Stanford, California, 1994.

[MBSU98] Z. Manna, A. Brown, H. B. Sipma, and T. E. Uribe. Visual abstractions for temporal verification. In *AMAST'98*. Lect. Notes in Comp. Sci., 1998.

[MP91a] Z. Manna and A. Pnueli. Completing the temporal picture. *Theor. Comp. Sci.*, 83(1):97–130, 1991.

[MP91b] Z. Manna and A. Pnueli. *The Temporal Logic of Reactive and Concurrent Systems: Specification.* Springer-Verlag, New York, 1991.

[MP94] Z. Manna and A. Pnueli. Temporal verification diagrams. In T. Ito and A. R. Meyer, editors, *Theoretical Aspects of Computer Software*, volume 789 of *Lect. Notes in Comp. Sci.*, pages 726–765. Springer-Verlag, 1994.

[MP95] Z. Manna and A. Pnueli. *Temporal Verification of Reactive Systems: Safety.* Springer-Verlag, New York, 1995.

[MW84] Z. Manna and P. Wolper. Synthesis of communicating processes from temporal logic specifications. *ACM Trans. Prog. Lang. Sys.*, 6:68–93, 1984.

[Pnu81] A. Pnueli. The temporal semantics of concurrent programs. *Theoretical Computer Science*, 13:1–20, 1981.

[SdRG89] F.A. Stomp, W.-P. de Roever, and R.T. Gerth. The μ-calculus as an assertion language for fairness arguments. *Inf. and Comp.*, 82:278–322, 1989.

[SUM99] H.B. Sipma, T.E. Uribe, and Z. Manna. Deductive model checking. *Formal Methods in System Design*, 15(1):49–74, 1999.

[Uri99] T. E. Uribe. *Abstraction-Based Deductive-Algorithmic Verification of Reactive Systems.* PhD thesis, Stanford University, 1999.

[Var91] M. Y. Vardi. Verification of concurrent programs – the automata-theoretic framework. *Annals of Pure Applied Logic*, 51:79–98, 1991.

[VW94] Moshe Y. Vardi and Pierre Wolper. Reasoning about infinite computations. *Inf. and Cont.*, 115(1):1–37, 1994.