Rules for Abstraction

Stephan Merz*

Institut für Informatik, Universität München
merz@informatik.uni-muenchen.de

Abstract. Abstraction techniques for the verification of reactive systems promise to provide a theoretical basis for the integration of automatic and interactive proof techniques. In this paper, we give an account of homomorphic abstraction by studying a series of proof rules in Lamport's Temporal Logic of Actions. We believe that the main advantage of a logical formalization of abstraction is that it points towards more refined abstraction techniques. Specifically, we demonstrate two novel techniques that appear helpful in the verification of liveness properties over abstract models.

1 Verification by Abstraction

Machine assistance in the verification of reactive systems traditionally comes in two different forms: interactive theorem proving allows the verification of arbitrarily complex systems, but it requires a large amount of training and can be tedious, even with the help of sophisticated tactics. Model checking and similar techniques, on the other hand, integrate smoothly with traditional testing and debugging and are largely automatic, but they are usually limited to systems whose state space is finite and relatively small.

Abstraction techniques [5, 7, 10, 12, 16, 18, 22, 23] promise to integrate the two approaches: in a first step, the original system model Φ is reduced to a smaller model $\overline{\Phi}$. If Φ is large or infinite, this step will in general require interactive proof techniques. In a second step, the small system $\overline{\Phi}$ is analyzed using automatic tools.

The reduction from Φ to $\overline{\Phi}$ is based on the structure of Φ. For example, in an automata-based approach, one would compute the abstract automaton $\overline{\Phi}$ by abstracting the initial condition of Φ and each possible state transition. However, fairness conditions imposed on Φ are not as easily abstracted, and liveness properties are often considered hard to prove by abstraction.

The contribution of this paper is twofold: first, we re-examine the principles that underly abstraction techniques by stating a series of proof rules in Lamport's Temporal Logic of Actions TLA [14]. Although our exposition can be easily rephrased in terms of other formalisms, this choice is convenient because TLA distinguishes between the level of actions and temporal formulas, and these two

* This work was supported by a grant from the state of Bavaria (Bayerischer Habilitations-Förderpreis).

levels are respectively associated with automatic and interactive theorem proving in proofs by abstraction. The second contribution is a suggestion to generalize known abstraction techniques by making use of standard laws of temporal logic. In this way, it is possible to go beyond purely structural abstractions and consider entire runs of a system rather than just individual state transitions. We present two specific novel techniques that we have found useful for the verification of liveness properties by abstraction. These techniques allow much of the tedium usually associated with hand proofs (or interactive proofs) of liveness properties to be left to the model checker. Finally, our approach allows the principle of abstraction to be itself formalized within a proof system, rather than to be stated as a separate meta-rule.

The outline of this paper is as follows: Section 2 gives a short overview of TLA. Section 3 introduces the principal proof rules for abstraction, which are applied to an example in section 4. Section 5 discusses techniques to obtain better abstractions. Finally, section 6 summarizes the key points and compares with existing work.

2 The Temporal Logic of Actions

We begin with a short review of the logic TLA, concentrating on the concepts that we use in the remainder of this paper. TLA is defined more formally in [14, 2]. Specifications and properties of systems are expressed in TLA as *temporal formulas*, which are interpreted over infinite sequences of states. States assign values to state variables that represent individual system components.

Syntactically, TLA is a two-tiered formalism whose temporal formulas are defined on top of *actions* (transition formulas), interpreted over pairs of states, that describe state transitions. This separation of the transition level from the temporal level is deliberate; TLA proof rules aim at reducing temporal properties to action reasoning. The syntax of actions is that of standard first-order formulas, except that state variables may have primed as well as standard, unprimed occurrences, with the convention that a primed occurrence like x' refers to the value of x in the second state of the state pair. For example, the action $x' = x + 1$ is true of a pair $\langle s, t \rangle$ of states if the value of x at state t equals the value of x at s plus one. If v is a variable, we write $\text{UNCHANGED}\, v$ for the action $v' = v$, and we use a similar convention for tuples of variables.

A *state predicate* is a transition formula that does not contain free primed variables; similarly, a *state function* is a term that contains only unprimed variables. If A is an action and v is a state function, then $[A]_v$ and $\langle A \rangle_v$ are abbreviations for $A \vee (v' = v)$ and $A \wedge (v' \neq v)$, where v' is obtained from v by priming all state variables. The state predicate $\text{ENABLED}\ A$ is an abbreviation for the formula $\exists x'_1, \ldots, x'_n : A$, where $x'_1, \ldots, x'_n$ are all primed variables that occur free in A. Semantically, $\text{ENABLED}\ A$ holds of state s iff there exists some state t such that A holds of the pair $\langle s, t \rangle$.

Temporal formulas are built from transition formulas using the connectives $\Box$ and $\boldsymbol{\exists}$ in addition to ordinary first-order connectives. The $\Box$ operator is the usual "always" operator of temporal logic. The quantifier $\exists$ represents hiding; the formula $\exists x : F$ is true of a state sequence σ if there exists some state sequence τ such that F holds of τ and where all states of τ agree with the corresponding states of σ, except possibly for the valuation of the variable x. Standard abbreviations include $\Diamond F$ for $\neg\Box\neg F$ and weak and strong fairness, defined as

$$\begin{aligned} \mathrm{WF}_v(A) &\triangleq \Box\Diamond\neg \textsc{Enabled}\ \langle A\rangle_v \vee \Box\Diamond\langle A\rangle_v \\ \mathrm{SF}_v(A) &\triangleq \Diamond\Box\neg \textsc{Enabled}\ \langle A\rangle_v \vee \Box\Diamond\langle A\rangle_v \end{aligned}$$

One distinguishing feature of TLA is that all temporal formulas are invariant under stuttering, that is, finite repetitions of identical states. For this reason, temporal formulas are restricted so that $\Box A$ is well-formed for an action A only if A is either a state predicate or if it is of the form $[B]_v$; the logic without this restriction is called "raw TLA" (RTLA) in [14]. Lamport [13] emphasizes the importance of stuttering-invariant specifications in the light of refinement. However, for the purposes of verification, it is often easier to work in the simpler logic RTLA, and we will do so in this paper. RTLA and TLA differ subtly in the precise semantics of the hiding operator $\exists$. Because our rules do not make essential use of quantification, they are sound for both versions of TLA.

3 Proof Rules for Abstraction

In this section we reconstruct the standard theory of verification by homomorphic abstraction in the context of TLA, introducing basic concepts and notations. Extensions of the standard theory will be presented in section 5.

3.1 Basic Principle

The overall principle that underlies homomorphic abstraction is to deduce some property P of a "concrete" system Φ from a proof of property Q for an "abstract" system $\overline{\Phi}$. The relation between the concrete and abstract state spaces is provided by an abstraction function h.

In TLA, specifications and properties are both represented by formulas, and implementation is implication. The state components of the concrete and abstract systems will be represented by different (tuples of) variables, say x for the concrete level and $\bar{x}$ for the abstract level, whose values at every state are related by the function h. (Variables that occur at both the abstract and the concrete level are implicitly left unabstracted.) The principle of abstraction proofs is reflected in the TLA proof rule

$$\frac{\begin{aligned} \Phi \wedge \Box A \wedge \Box(\bar{x} = h(x)) &\Rightarrow \overline{\Phi} \\ \overline{\Phi} &\Rightarrow Q \\ Q \wedge \Box A \wedge \Box(\bar{x} = h(x)) &\Rightarrow P \end{aligned}}{\Phi \wedge \Box A \Rightarrow P} \quad (\bar{x} \notin \mathit{Vars}(\Phi, A, P)) \tag{1}$$

The second premise of rule (1) asserts that the abstract system $\overline{\Phi}$ satisfies the abstract property Q. The first and third premises relate the abstract and concrete levels of description and assert that the abstract specification $\overline{\Phi}$ is an image of the concrete specification Φ under the function h, and that the concrete property P is a pre-image under h of the abstract property Q. The auxiliary formula $\Box A$, where A is a transition formula, can be used to represent the "context" of the concrete system. Typically, A will imply the next-state relation and pre-established invariants of the concrete system. Whereas rule (1) remains correct if all occurrences of $\Box A$ are erased, a suitable choice of A is usually essential to make the rule applicable in practice.

To prove the correctness of rule (1), we observe that its premises and propositional logic imply

$$\Phi \wedge \Box A \wedge \Box(\bar{x} = h(x)) \;\Rightarrow\; P$$

By standard quantifier rules and using the assumption that none of the abstract variables $\bar{x}$ occur in the concrete-level formulas Φ, A or P, we obtain

$$\Phi \wedge \Box A \wedge (\exists\, \bar{x} : \Box(\bar{x} = h(x))) \;\Rightarrow\; P$$

Finally, the conclusion of rule (1) follows from the tautology $\Box(h(x) = h(x))$, which implies the validity of the conjunct $\exists\, \bar{x} : \Box(\bar{x} = h(x))$.

3.2 Generic Abstractions

The second premise $\overline{\Phi} \Rightarrow Q$ of rule (1) concerns only the abstract system, and we expect to use automatic verification techniques in its proof. However, the other premises are also formulated as proof obligations in temporal logic, and it is not clear why they should be easier to prove than the conclusion $\Phi \Rightarrow P$ itself. It is here where we need to use the idea that the abstract and concrete levels are structurally related by the homomorphism h. In a logical setting, this means that the formulas at the two levels have similar shape.

Rule (1) suggests that it makes sense to define *two* kinds of abstractions: the abstract-level formula $\overline{\Phi}$ is weaker than its concrete-level counterpart Φ, whereas Q is stronger than P. Actually, this setup is a little more general than the definitions that are often found in the literature, which consider only weakening abstractions and require the properties P and Q at the abstract and concrete level to be identical. We believe that this added bit of generality can be useful and comes at no extra price.

Definition 1. Let h be a state function and A be an action.

1. For any action N, an action $\overline{N}^+$ (resp., $\overline{N}^-$) is called a *weakening (resp., strengthening) abstraction* of N w.r.t. h and A if the following formulas are valid:

$$A \wedge (\bar{x} = h(x)) \wedge (\bar{x}' = h(x)') \;\Rightarrow\; (N \Rightarrow \overline{N}^+) \quad \text{resp.}$$
$$A \wedge (\bar{x} = h(x)) \wedge (\bar{x}' = h(x)') \;\Rightarrow\; (\overline{N}^- \Rightarrow N)$$

2. For any temporal formula F, a temporal formula $\overline{F}^+$ (resp., $\overline{F}^-$) is called a *weakening (resp., strengthening) abstraction* of F w.r.t. h and A if the following formulas are valid:

$$\begin{array}{l} \Box A \wedge \Box(\bar{x} = h(x)) \;\Rightarrow\; (F \Rightarrow \overline{F}^+) \quad \text{resp.} \\ \Box A \wedge \Box(\bar{x} = h(x)) \;\Rightarrow\; (\overline{F}^- \Rightarrow F) \end{array}$$

Note that TRUE and FALSE are trivial abstractions of any formula. In general, a suitable choice of the auxiliary action A will be important to find good abstractions. Note also that there can be different abstractions of a given formula G, although the values at the concrete and abstract levels are functionally related. Therefore, our definition of a weakening abstraction subsumes the notion of "abstraction" as well as that of "approximation" from [5].

Using the notation of definition 1, we can rephrase rule (1), assuming that $\overline{\Phi}^+$ and $\overline{P}^-$ are weakening and strengthening abstractions of Φ and P:

$$\frac{\overline{\Phi}^+ \Rightarrow \overline{P}^-}{\Phi \Rightarrow P} \tag{2}$$

The idea of a homomorphic relationship between the concrete and abstract specifications implies that abstractions of complex systems can be obtained from abstractions of their components. This is made precise in the following theorem, which is proved by induction on the structure of temporal formulas.

Theorem 2. *Let F be a temporal or transition formula built from elementary actions R_i by the connectives of TLA such that no variables that occur free in A, $\bar{x} = h(x)$ or $\bar{x}' = h(x)'$ occur bound in F, and assume given weakening and strengthening abstractions $\overline{R_i}^+$ and $\overline{R_i}^-$ w.r.t. h and A for every R_i.*

1. *$\overline{F}^+$ is a weakening abstraction of F, where $\overline{F}^+$ denotes the formula obtained from F by replacing every positive (resp., negative) occurrence of R_i by $\overline{R_i}^+$ (resp., $\overline{R_i}^-$).*
2. *$\overline{F}^-$ is a strengthening abstraction of F, where $\overline{F}^-$ denotes the formula obtained from F by replacing every positive (resp., negative) occurrence of R_i by $\overline{R_i}^-$ (resp., $\overline{R_i}^+$).*

Theorem 2 asserts that one can build abstractions of complex formulas F given abstractions of elementary transition formulas. The abstractions obtained in this way have the same shape as F, except that the concrete actions R_i are replaced by their abstract counterparts $\overline{R_i}^+$ or $\overline{R_i}^-$. The decision which transition formulas should be regarded as elementary is left to the user; typically, one might consider the initial conditions and the individual actions of single processes as elementary.

Theorem 2 does not imply that ENABLED $\overline{G}$ is an abstraction of ENABLED G if $\overline{G}$ is an abstraction of G, because the ENABLED operator is defined by quantification over state variables. For example, FALSE is a weakening abstraction of G with respect to h and $\neg G$, but since $\neg G$ does not imply $\neg$ENABLED G, we may

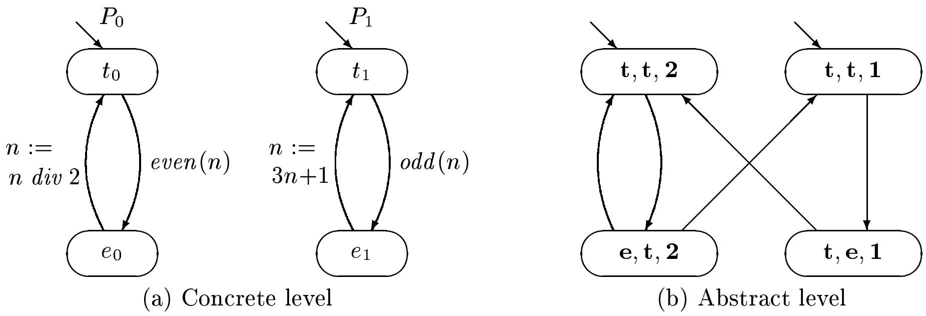

Fig. 1. The dining mathematicians

not conclude that ENABLED FALSE (which is again FALSE) is a weakening abstraction of ENABLED G w.r.t. $\neg G$. One can show that abstraction can be lifted over ENABLED provided that $A \wedge$ ENABLED G implies ENABLED $(G \wedge A \wedge (\bar{x}' = h(x)'))$. Although this side condition is frequently satisfied, it may be more practical to replace enabledness conditions by equivalent quantifier-free state predicates. This complication in abstracting enabledness predicates affects the abstract representation of concrete-level fairness conditions. Besides, A has both negative and positive occurrences in the formulas $\mathrm{WF}_v(A)$ and $\mathrm{SF}_v(A)$, and one would therefore have to find an abstraction $\overline{A}$ of A that is both weakening and strengthening in order to represent the concrete fairness conditions as $\mathrm{WF}_{\overline{v}}(\overline{A})$ or $\mathrm{SF}_{\overline{v}}(\overline{A})$. We will discuss issues of fairness and liveness under abstractions in section 5.

Rule (2) and theorem 2 suggest a methodology for verification by abstraction where automated verification techniques are used to reason about the abstract system and establish the premise of rule (2), whereas first-order theorem proving is used to verify abstractions of elementary transition formulas. Because the latter proof obligations do not involve temporal logic, their proof may benefit from well-developed theories for interactive theorem provers such as PVS [20] or Isabelle [21].

4 An Example

Let us illustrate our theory at the hand of a trivial example, the so-called "dining mathematicians" problem introduced in [7]. Figure 1(a) depicts the system, which consists of two processes P_0 and P_1 that communicate via the shared variable n. Both processes have a "thinking" state t_i and an "eating" state e_i; initially they are thinking. The variable n is initialized to some unspecified positive natural number. The purpose of the system is to ensure mutually exclusive access to the dining room for both processes without starvation. Process P_0 may eat only if n is even and it divides n by 2 when it resumes thinking. Process P_1 may eat only if n is odd and sets n to $3n+1$ when it leaves the dining room.

Figure 2 contains a TLA specification of the dining mathematicians program, represented by the temporal formula *DM*. We wish to verify the following

$$
\begin{array}{rcl}
\mathit{Init} & \triangleq & (n > 0) \wedge (c_0 = \mathbf{t}) \wedge (c_1 = \mathbf{t}) \\
\mathit{Eat}_0 & \triangleq & (c_0 = \mathbf{t}) \wedge \mathit{even}(n) \wedge (c_0' = \mathbf{e}) \wedge \text{UNCHANGED}\ \langle n, c_1 \rangle \\
\mathit{Think}_0 & \triangleq & (c_0 = \mathbf{e}) \wedge (c_0' = \mathbf{t}) \wedge (n' = n\ \mathit{div}\ 2) \wedge \text{UNCHANGED}\ c_1 \\
\mathit{Eat}_1 & \triangleq & (c_1 = \mathbf{t}) \wedge \mathit{odd}(n) \wedge (c_1' = \mathbf{e}) \wedge \text{UNCHANGED}\ \langle n, c_0 \rangle \\
\mathit{Think}_1 & \triangleq & (c_1 = \mathbf{e}) \wedge (c_1' = \mathbf{t}) \wedge (n' = 3 * n + 1) \wedge \text{UNCHANGED}\ c_0 \\
\mathit{Next} & \triangleq & \mathit{Eat}_0 \vee \mathit{Think}_0 \vee \mathit{Eat}_1 \vee \mathit{Think}_1 \\
v & \triangleq & \langle c_0, c_1, n \rangle \\
\mathit{DM} & \triangleq & \mathit{Init} \wedge \Box[\mathit{Next}]_v \wedge \Box\Diamond\langle \mathit{Next} \rangle_v
\end{array}
$$

Fig. 2. TLA specification of the dining mathematicians program

properties:

$$
\begin{array}{llll}
(\mathit{Pos}) & \mathit{DM} \Rightarrow \Box(n \in \mathbb{N}^+) & (\mathit{Excl}) & \mathit{DM} \Rightarrow \Box\neg(c_0 = \mathbf{e} \wedge c_1 = \mathbf{e}) \\
(\mathit{Live}_0) & \mathit{DM} \Rightarrow \Box\Diamond(c_0 = \mathbf{e}) & (\mathit{Live}_1) & \mathit{DM} \Rightarrow \Box\Diamond(c_1 = \mathbf{e})
\end{array}
$$

The invariance properties (Pos) and (Excl) assert that the shared variable n is a positive natural number[1] throughout any execution of the system specified by DM, and that the two processes never meet in the dining room. The liveness properties (Live_0) and (Live_1) assert starvation freedom for the two processes. We wish to establish all four properties by abstraction, following the method outlined in section 3. We replace the integer variable n of the concrete program by an abstract variable $\bar{n}$ that takes values from the set $\{\mathbf{0}, \mathbf{1}, \mathbf{2}\}$ where $\mathbf{0}$ represents the natural number 0, and $\mathbf{1}$ and $\mathbf{2}$ represent any odd or even positive natural number. Formally, the abstraction function h is defined as

$$
h(n) \triangleq \textbf{if } n = 0 \textbf{ then } \mathbf{0} \textbf{ elsif } \mathit{odd}(n) \textbf{ then } \mathbf{1} \textbf{ else } \mathbf{2}
$$

We let the conjunction of the specification's next-state relation $[\mathit{Next}]_v$ and the predicate $n \in \mathbb{N}$ be the auxiliary transition formula A that defines the context of the abstraction. The theorem $\mathit{DM} \Rightarrow \Box(n \in \mathbb{N})$, which asserts the type correctness of program DM, can be established by a straightforward invariance proof, or even automatically by the type checker [15] of a typical interactive theorem prover.

Figure 3 contains an abstract version $\overline{\mathit{DM}}$ of the dining mathematicians program[2]. It has the same shape as the specification DM of figure 2, except that the elementary transition formulas have been replaced by abstract versions. Using an interactive theorem prover, these abstract formulas are quickly proved to be weakening abstractions[3] of their concrete counterparts with respect to h and A.

---

[1] We let $\mathbb{N}$ denote the set $\{0, 1, 2, \ldots\}$ of natural numbers and $\mathbb{N}^+$ denote the set $\mathbb{N} \setminus \{0\}$.

[2] We adopt the convention of writing multi-line conjunctions or disjunctions as lists bulleted with the connectives $\wedge$ and $\vee$.

[3] In this simple example, they are also strengthening abstractions of the concrete formulas (because the next-state relation is a conjunct of A), but this fact is not used in the verification.

$$
\begin{array}{rcl}
\overline{Init} & \triangleq & (\bar{n} \in \{\mathbf{1},\mathbf{2}\}) \wedge (c_0 = \mathbf{t}) \wedge (c_1 = \mathbf{t}) \\
\overline{Eat_0} & \triangleq & (c_0 = \mathbf{t}) \wedge (\bar{n} \in \{\mathbf{0},\mathbf{2}\}) \wedge (c_0' = \mathbf{e}) \wedge \text{UNCHANGED } \langle \bar{n}, c_1 \rangle \\
\overline{Think_0} & \triangleq & \wedge\ (c_0 = \mathbf{e}) \wedge (c_0' = \mathbf{t}) \wedge \text{UNCHANGED } c_1 \\
 & & \wedge \vee\ (\bar{n} = \mathbf{0}) \wedge (\bar{n}' = \mathbf{0}) \\
 & & \quad \vee\ (\bar{n} = \mathbf{1}) \wedge (\bar{n}' \in \{\mathbf{0},\mathbf{1},\mathbf{2}\}) \\
 & & \quad \vee\ (\bar{n} = \mathbf{2}) \wedge (\bar{n}' \in \{\mathbf{1},\mathbf{2}\}) \\
\overline{Eat_1} & \triangleq & (c_1 = \mathbf{t}) \wedge (\bar{n} = \mathbf{1}) \wedge (c_1' = \mathbf{e}) \wedge \text{UNCHANGED } \langle \bar{n}, c_0 \rangle \\
\overline{Think_1} & \triangleq & \wedge\ (c_1 = \mathbf{e}) \wedge (c_1' = \mathbf{t}) \wedge \text{UNCHANGED } c_0 \\
 & & \wedge \vee\ (\bar{n} \in \{\mathbf{0},\mathbf{2}\}) \wedge (\bar{n}' = \mathbf{1}) \\
 & & \quad \vee\ (\bar{n} = \mathbf{1}) \wedge (\bar{n}' = \mathbf{2}) \\
\overline{Next} & \triangleq & \overline{Eat_0} \vee \overline{Think_0} \vee \overline{Eat_1} \vee \overline{Think_1} \\
\bar{v} & \triangleq & \langle c_0, c_1, \bar{n} \rangle \\
\overline{DM} & \triangleq & \overline{Init} \wedge \Box[\overline{Next}]_{\bar{v}} \wedge \Box\Diamond\langle\overline{Next}\rangle_{\bar{v}}
\end{array}
$$

Fig. 3. Abstract version of the dining mathematicians program

For example, the proof of

$$[Next]_v \wedge (n \in \mathbb{N}) \wedge (\bar{n} = h(n)) \wedge (\bar{n}' = h(n)') \Rightarrow (Think_1 \Rightarrow \overline{Think_1})$$

relies on the fact that $3n+1$ is odd if n is an even integer and even otherwise. By theorem 2, it follows that $\overline{DM}$ is a weakening abstraction of DM.

Specification $\overline{DM}$ describes a finite-state system that can be analyzed by conventional model checkers. Its reachable state space is depicted in figure 1(b), where each state is labelled with the values of the state variables c_0, c_1, and $\bar{n}$. The following properties are easily read off the state transition diagram:

$$
\begin{array}{llll}
(\overline{Pos}) & \overline{DM} \Rightarrow \Box(\bar{n} \in \{\mathbf{1},\mathbf{2}\}) & \qquad (\overline{Excl}) & \overline{DM} \Rightarrow \Box\neg(c_0 = \mathbf{e} \wedge c_1 = \mathbf{e}) \\
(\overline{Live_0}) & \overline{DM} \Rightarrow \Box\Diamond(c_0 = \mathbf{e}) & &
\end{array}
$$

From these properties and rule (2), we may conclude that properties Pos, $Excl$, and $Live_0$ hold for the concrete specification DM. For example, $\bar{n} \in \{\mathbf{1},\mathbf{2}\}$ is a strengthening abstraction of $n \in \mathbb{N}^+$, and therefore $\Box(\bar{n} \in \{\mathbf{1},\mathbf{2}\})$ is a strengthening abstraction of $\Box(n \in \mathbb{N}^+)$.

Unfortunately, we cannot prove $\overline{DM} \Rightarrow \Box\Diamond(c_1 = \mathbf{e})$, which would imply property $Live_1$ of starvation-freedom for process P_1, because of the cycle that connects the two left-hand states of figure 1(b). In fact, no finite-state homomorphic abstraction of specification DM allows a proof of property $Live_1$, because for every k there exists an execution of DM where P_0 eats k times in succession while P_1 has to wait. Techniques that allow homomorphic abstractions to be systematically improved, and which in particular allow a proof of property $Live_1$, will be discussed in section 5.

Let us emphasize that this example serves only as an illustration of the concepts of section 3. Automatic provers that employ techniques for invariant generation based on linear arithmetic [3, 4] can verify the three properties (Pos), ($Excl$), and ($Live_0$) without explicitly constructing an abstract version of DM.

5 Beyond Structural Abstractions

A useful abstraction function maps distinct concrete states to a single abstract state. This desired reduction of the size of the state space, however, does not come for free: first, it is likely to introduce cycles in the state graph of the abstract system that have no counterpart in the concrete system. Second, the concrete specification may contain subformulas that are too "fine-grained" and admit no useful abstraction. Technically, it may be impossible to find nontrivial strengthening abstractions for negative subformulas of the specification, such as the enabling conditions in fairness constraints. Both problems particularly affect the verification of liveness properties. In the first case, one can strengthen the abstract specification. In the second case, it may be helpful to weaken the concrete specification. Because TLA does not distinguish between specifications and properties, we may formally express both principles by the rule

$$\frac{\Phi \Rightarrow H \qquad \overline{\Phi}^+ \wedge \overline{H}^+ \Rightarrow \overline{P}^-}{\Phi \Rightarrow P} \tag{3}$$

In this rule, the temporal formula H, which represents either an auxiliary property or a weaker specification, is implied by the concrete specification Φ, and $\overline{P}^-$ is deduced from $\overline{\Phi}^+ \wedge \overline{H}^+$ instead of just $\overline{\Phi}^+$ as in (2). According to the general methodology of abstraction proofs, the premise $\Phi \Rightarrow H$ should be reduced to transition-level proof obligations by general principles. We will discuss two instances of rule (3) that we have found useful: one technique eliminates "false" cycles and the other one attempts to weaken fairness conditions in order to obtain stronger abstractions.

5.1 Strengthening by Well-Founded Orderings

If cycles in the abstract state graph cannot be broken with the help of a better abstraction function, it may still be possible to strengthen the abstract specification using an auxiliary liveness property of the concrete system. Liveness proofs are usually based on induction over a well-founded ordering $(S, \prec)$, a concept formalized in standard libraries of interactive theorem provers. The TLA formula

$$\Box(y \in S) \Rightarrow (\Box\Diamond\langle y' \prec y\rangle_y \Rightarrow \Box\Diamond\langle y' \not\prec y\rangle_y) \tag{4}$$

asserts that if y is always an element of S and if the value of y decreases infinitely often, then there have to be infinitely many steps that change y without decreasing it. If $\prec$ is a linear ordering, $\langle y' \not\prec y\rangle_y$ is of course equivalent to $\langle y \prec y'\rangle_y$. Formula (4) is valid if $(S, \prec)$ is well-founded. If $\overline{Decr}^-$ and $\overline{NonDecr}^+$ are strenghtening and weakening abstractions, respectively, of $\langle y' \prec y\rangle_y$ and $\langle y \not\prec y'\rangle_y$ with respect to h and $A \wedge (y \in S)$, we may use (4) and theorem 2 to obtain the instance

$$\frac{\Phi \Rightarrow \Box(y \in S) \qquad \overline{\Phi}^+ \wedge (\Box\Diamond\overline{Decr}^- \Rightarrow \Box\Diamond\overline{NonDecr}^+) \Rightarrow \overline{P}^-}{\Phi \Rightarrow P} \tag{5}$$

of rule (3). The first premise of (5) asserts an invariant of Φ that may be proved by standard invariant rules (perhaps even by type checking) or by another proof by abstraction. The second premise strengthens the abstract specification $\overline{\Phi}^+$ in order to establish the desired property.

We demonstrate rule (5) at the hand of the remaining liveness property $Live_1$ from the dining mathematicians example of section 4, choosing the well-founded ordering $(\mathbb{N}^+, <)$. We have already established the first premise of rule (5) as property *Pos*. Moreover, the only actions that change n are $Think_0$ and $Think_1$, and in fact $\overline{Think_0}$ and $\overline{Think_1}$ are strengthening and weakening abstractions, respectively, of $\langle n' < n \rangle_n$ and $\langle n < n' \rangle_n$ with respect to h and $A \wedge (n \in \mathbb{N}^+)$. Hence, to prove $Live_1$, it suffices to show

$$\overline{DM} \wedge (\Box\Diamond \overline{Think_0} \Rightarrow \Box\Diamond \overline{Think_1}) \Rightarrow \Box\Diamond(c_1 = \mathbf{e}) \tag{6}$$

The additional conjunct prohibits the abstract system to persistently cycle between the two left-hand states of figure 1(b), because any such execution would perform infinitely many $\overline{Think_0}$ actions, but only finitely many $\overline{Think_1}$ actions. In fact, decision procedures for linear-time temporal logic such as the model checking component of STeP [4] easily verify property (6).

5.2 Weakening Fairness

The lifting of abstractions to complex formulas as suggested by theorem 2 is a simple syntactic procedure and is therefore sensitive to the syntactic shape of a formula. If the concrete specification Φ contains both negative and positive occurrences of elementary subformulas, the lifting operation is not monotonic: a weaker formula at the concrete level may have a stronger abstraction with respect to a given mapping. Because enabledness conditions occur negatively in fairness conditions, it may be useful to weaken the fairness conditions of the concrete specification before constructing the abstraction.

Many specifications have an infinite state space because they are generic in some parameter. There will often be a fairness condition for each possible parameter value, that is, the TLA specification Φ contains conjuncts of the form $\forall x \in S : \mathrm{XF}_v(A(x))$, where XF denotes either SF or WF. Now suppose that we wish to verify a property of the form $\Phi \Rightarrow \forall x \in S : P(x)$. The first step in the proof is to introduce a fresh Skolem constant, say $\mathbf{x}$, and prove $\Phi \wedge (\mathbf{x} \in S) \Rightarrow P(\mathbf{x})$. Because of interaction between the subprocesses of the system, it will be impossible to prove this property from the process associated with $\mathbf{x}$ alone, so we have to distinguish between actions that concern $\mathbf{x}$ and actions of different processes. For the sake of simplicity, assume that it is not necessary to distinguish between individual parameter values other than $\mathbf{x}$, but that it is enough to consider the two-element abstract parameter domain that consists of $\mathbf{x}$ and a second constant $\mathbf{y}$ representing an arbitrary value different from $\mathbf{x}$.

An application of theorem 2 to compute abstractions for the fairness conditions of Φ requires strengthening abstractions of $\textsc{Enabled}\ \langle A(y) \rangle_v$ for $y \in S \backslash \{\mathbf{x}\}$.

$$
\begin{array}{rcl}
\mathit{Init} & \triangleq & \forall x \in P : \neg \mathit{req}[x] \wedge \neg \mathit{granted}[x] \\
R(x) & \triangleq & \mathit{req}'(x) \wedge \forall y \in P \setminus \{x\} : \text{UNCHANGED } \langle \mathit{req}[y], \mathit{granted} \rangle \\
G(x) & \triangleq & \wedge\ \mathit{req}(x) \wedge \text{UNCHANGED } \mathit{req} \\
 & & \wedge\ \forall y \in P : \neg \mathit{granted}[y] \wedge (\mathit{granted}'[y] \Leftrightarrow (y = x)) \\
F(x) & \triangleq & \wedge\ \mathit{granted}[x] \wedge \neg \mathit{granted}'[x] \wedge \neg \mathit{req}'[x] \\
 & & \wedge\ \forall y \in P \setminus \{x\} : \text{UNCHANGED } \langle \mathit{req}[y], \mathit{granted}[y] \rangle \\
v & \triangleq & \langle \mathit{req}, \mathit{granted} \rangle \\
\mathit{RM} & \triangleq & \wedge\ \mathit{Init} \wedge \Box[\exists x \in P : R(x) \vee G(x) \vee F(x)]_v \\
 & & \wedge\ \forall x \in P : \mathrm{WF}_v(F(x)) \wedge \mathrm{SF}_v(G(x))
\end{array}
$$

Fig. 4. A simple resource manager

Unfortunately, we can no longer identify individual processes different from $\mathbf{x}$ in the abstract state space, and can therefore not compute a useful strengthening abstraction of these enabledness condition. Instead, we will have to use the trivial abstraction TRUE for the fairness conditions, and are likely to introduce deadlocks in the abstract specification. In such a situation we may use the TLA rule

$$
\frac{\begin{array}{l} N \wedge \neg(\exists z \in M : \langle A(z) \rangle_v) \wedge (y \in M) \\ \quad \Rightarrow (\text{Enabled } \langle A(y) \rangle_v \Rightarrow (\text{Enabled } \langle A(y) \rangle_v)') \end{array}}{\Box N \wedge \forall y \in M : \mathrm{XF}_v(A(y)) \Rightarrow \mathrm{XF}_v(\exists y \in M : A(y))} \tag{7}
$$

Rule (7) asserts that it is weaker to require fairness with respect to *some* action $A(y)$ than to require fairness of $A(y)$ for each individual value of y, provided that no action except $A(z)$ for some value of z can disable $A(y)$. It is therefore enough to find abstractions for $\text{Enabled}(\exists y \in S \setminus \{\mathbf{x}\} : A(y))$, that is, parameter values that map to the same abstract element need no longer be distinguished.

As a concrete, if trivial, example, consider the specification of a shared-variable resource manager shown in figure 4. The state of the system is encoded by two arrays *req* and *granted* indexed by a set P of processes. With each process are associated three actions: $R(x)$ and $F(x)$ represents process actions requesting and releasing the resource. The action $G(x)$ models the granting of the resource by the manager. Although the specification itself is overly simplistic, its syntactic shape is typical of shared-variable specifications.

Suppose that we wish to prove liveness for each process $x \in P$, which is expressed by the formula $\mathit{RM} \Rightarrow \forall x \in P : \Box(\mathit{req}[x] \Rightarrow \Diamond \mathit{granted}[x])$. As indicated above, we introduce a fresh Skolem constant $\mathbf{x}$ and try to prove

$$
\mathit{RM} \wedge (\mathbf{x} \in P) \Rightarrow \Box(\mathit{req}[\mathbf{x}] \Rightarrow \Diamond \mathit{granted}[\mathbf{x}]) \tag{8}
$$

By (7), whose antecedent (for both the F and the G actions) is easily seen to be satisfied, we may weaken RM to the specification

$$
\begin{array}{l}
\wedge\ \mathit{Init} \\
\wedge\ \Box[R(\mathbf{x}) \vee G(\mathbf{x}) \vee F(\mathbf{x}) \vee \exists y \in P \setminus \{\mathbf{x}\} : R(y) \vee G(y) \vee F(y)]_v \\
\wedge\ \mathrm{WF}_v(F(\mathbf{x})) \wedge \mathrm{SF}_v(G(\mathbf{x})) \\
\wedge\ \mathrm{WF}_v(\exists y \in P \setminus \{\mathbf{x}\} : F(y)) \wedge \mathrm{SF}_v(\exists y \in P \setminus \{\mathbf{x}\} : G(y))
\end{array}
$$

Applying theorem 2 to this weaker specification, we arrive at the abstract specification

$$\begin{array}{l} \wedge\ \neg req[\mathbf{x}] \wedge \neg req[\mathbf{y}] \wedge \neg granted[\mathbf{x}] \wedge \neg granted[\mathbf{y}] \\ \wedge\ \Box[R(\mathbf{x}) \vee G(\mathbf{x}) \vee F(\mathbf{x}) \vee R(\mathbf{y}) \vee G(\mathbf{y}) \vee F(\mathbf{y})]_v \\ \wedge\ \mathrm{WF}_v(F(\mathbf{x})) \wedge \mathrm{SF}_v(G(\mathbf{x})) \wedge \mathrm{WF}_v(F(\mathbf{y})) \wedge \mathrm{SF}_v(G(\mathbf{y})) \end{array}$$

from which (8) can be established automatically.

6 Conclusion

We have given a logical reconstruction of the principle of homomorphic abstraction, which has been widely investigated from both theoretical and practical points of view. The principle that underlies proofs by abstraction can be concisely expressed as a proof rule in temporal logic. For this to be useful in practice, it is essential that abstractions of complex systems can be composed from abstractions of their elementary constituents (theorem 2). In particular, one can build the abstraction from a syntactic description of the concrete program, without constructing its state graph.

The setting of temporal logic makes it natural to take entire system runs into account and to explore interactions between abstraction proofs and ordinary temporal reasoning. In this way, we are able to verify interesting liveness properties by abstraction. Nitsche and Wolper [19] have also addressed the preservation of liveness under abstractions and have obtained more specific results for a restricted class of homomorphisms.

We have used TLA in our exposition because its distinction between the transition and temporal levels of description is a good match for the methodology of abstraction that combines interactive theorem proving and model checking. Of course, our concrete results are not restricted to TLA. For example, rule (5) is easily reformulated in an automata-theoretic framework [23, 24], where one would add a Streett condition. Our rules also formalize the method suggested in [8] to strengthen abstract specifications by auxiliary properties in order to exclude unrealistic counter-examples reported by model checkers.

We have concentrated on homomorphic abstractions, the most widely considered abstraction method. It has been argued [7, 22] that it may be useful to consider more general relational instead of functional abstractions, possibly restricted to Galois connections as in the theory of abstract interpretation [6]. To some extent, this flexibility is already present in our approach because logical properties naturally form a lattice. The most useful generalization of our basic rule (1) in terms of an abstraction relation R seems to be the rule

$$\frac{\begin{array}{rcl} \Phi \wedge \Box A & \Rightarrow & \exists\, \bar{x} : \Box R(x, \bar{x}) \wedge \overline{\Phi} \\ \overline{\Phi} & \Rightarrow & Q \\ Q \wedge \Box A \wedge \Box R(x, \bar{x}) & \Rightarrow & P \end{array}}{\Phi \wedge \Box A \ \Rightarrow\ P} \quad \bar{x} \notin \mathit{Vars}(P) \tag{9}$$

One could use auxiliary variables [1] to prove the first premise of (9). It remains to explore whether the additional complexity is justified in practice.

Throughout our exposition, we have assumed that the necessary abstractions are provided by the user. A useful abstraction methodology also has to assist in the actual construction of the abstractions. Graf and Saidi [9] propose to exploit powerful automatic proof procedures of the interactive verifier to heuristically compute sound, although perhaps not optimal, abstractions, and to improve these as necessary.

We have used our approach to verify a novel reader-writer algorithm developed at Siemens Corporate Research [11]. In that case study, we have found abstraction particularly helpful to establish liveness properties, which are notoriously tedious to prove in temporal logic and are traditionally considered hard to verify by abstraction. The concepts presented in this paper have been implemented in an embedding of TLA in the theorem prover Isabelle [17]. We are investigating the integration of a tableau prover for linear temporal logic into our Isabelle embedding.

Acknowledgement. Comments by Martín Abadi, Leslie Lamport, Olaf Müller, and the referees on earlier drafts of this paper have helped to clarify the exposition.

References

1. Martín Abadi and Leslie Lamport. The existence of refinement mappings. *Theoretical Computer Science*, 81(2):253–284, May 1991.
2. Martín Abadi and Stephan Merz. On TLA as a logic. In Manfred Broy, editor, *Deductive Program Design*, NATO ASI series F, pages 235–272. Springer-Verlag, Berlin, 1996.
3. S. Bensalem, Y. Lakhnech, and H. Saidi. Powerful techniques for the automatic generation of invariants. In Rajeev Alur and Thomas A. Henzinger, editors, *Computer Aided Verification: 8th International Conference*, volume 1102 of *Lecture Notes in Computer Science*, pages 323–335. Springer-Verlag, 1996.
4. Nikolaj Bjørner, Anca Browne, Edward Chang, Michael Colón, Arjun Kapur, Zohar Manna, Henny B. Sipma, and Tomás E. Uribe. STeP: deductive-algorithmic verification of reactive and real-time systems. In Rajeev Alur and Thomas A. Henzinger, editors, *Computer Aided Verification: 8th International Conference*, volume 1102 of *Lecture Notes in Computer Science*, pages 415–418. Springer-Verlag, 1996.
5. Edmund M. Clarke, Orna Grumberg, and David E. Long. Model checking and abstraction. *ACM Transactions on Programming Languages and Systems*, 16(5):1512–1542, September 1994. A preliminary version appeared in POPL 1992.
6. Patrick Cousot and Radhia Cousot. Abstract interpretation: A unified lattice model for static analysis of programs by construction or approximation of fixpoints. In *4th ACM Symposium on Principles of Programming Languages*, pages 238–252, Los Angeles, California, 1977. ACM Press.
7. Dennis Dams, Orna Grumberg, and Rob Gerth. Abstract interpretation of reactive systems: Abstractions preserving $\forall$CTL*, $\exists$CTL* and CTL*. In Ernst-Rüdiger Olderog, editor, *Programming Concepts, Methods, and Calculi (PROCOMET '94)*, pages 561–581, Amsterdam, 1994. IFIP Transactions, North Holland/Elsevier.
8. J. Dingel and T. Filkorn. Model checking for infinite state systems using data abstraction, assumption-commitment style reasoning and theorem proving. In *Computer-Aided Verification*, volume 939 of *Lecture Notes in Computer Science*, Berlin, 1995. Springer-Verlag.

9. Susanne Graf and Hassan Saidi. Construction of abstract state graphs with PVS. In Orna Grumberg, editor, *Computer Aided Verification: 9th International Conference*, volume 1254 of *Lecture Notes in Computer Science*, pages 72–83. Springer-Verlag, 1997.
10. Klaus Havelund and Natarajan Shankar. Experiments in theorem proving and model checking for protocol verification. In Marie-Claude Gaudel and Jim Woodcock, editors, *Formal Methods Europe*, pages 662–681, Berlin, March 1996. Springer-Verlag.
11. Hermann Hellwagner. Scalable readers/writers synchronization on shared-memory machines. Technical report, Siemens AG, ZFE ST SN 2, 1993.
12. Robert P. Kurshan. Analysis of discrete event coordination. In J.W. de Bakker, W.-P. de Roever, and G. Rozenberg, editors, *Proceedings REX Workshop on Stepwise Refinement of Distributed Systems: Models, Formalisms, Correctness*, volume 430 of *Lecture Notes in Computer Science*, pages 414–454. Springer-Verlag, May 1989.
13. Leslie Lamport. What good is temporal logic? In R. E. A. Mason, editor, *Information Processing 83: Proceedings of the IFIP 9th World Congress*, pages 657–668, Paris, September 1983. IFIP, North-Holland.
14. Leslie Lamport. The Temporal Logic of Actions. *ACM Transactions on Programming Languages and Systems*, 16(3):872–923, May 1994.
15. Leslie Lamport and Lawrence C. Paulson. Should your specification language be typed? Research Report 147, Digital Equipment Corporation, Systems Research Center, Palo Alto, California, May 1997.
16. Claire Loiseaux, Susanne Graf, Joseph Sifakis, Ahmed Bouajjani, and Saddek Bensalem. Property preserving abstractions for the verification of concurrent systems. *Formal Methods in System Design*, 6:11–44, 1995. A preliminary version appeared as Spectre technical report RTC40, Grenoble, France, 1993.
17. Stephan Merz. Isabelle/TLA. Available on the WWW at URL http://www4.informatik.tu-muenchen.de/~merz/isabelle/, 1997.
18. Olaf Müller and Tobias Nipkow. Combining model checking and deduction for I/O-automata. In *Tools and Algorithms for the Construction and Analysis of Systems*, volume 1019 of *Lecture Notes in Computer Science*, pages 1–16. Springer-Verlag, 1995.
19. Ulrich Nitsche and Pierre Wolper. Relative liveness and behavior abstraction. In *Principles of Distributed Computing*. ACM Press, 1997. to appear.
20. Sam Owre, John Rushby, Natarajan Shankar, and Friedrich von Henke. Formal verification for fault-tolerant architectures: Prolegomena to the design of PVS. *IEEE Transactions on Software Engineering*, 21(2):107–125, February 1995.
21. Lawrence C. Paulson. *Isabelle: A Generic Theorem Prover*, volume 828 of *Lecture Notes in Computer Science*. Springer-Verlag, Berlin, Heidelberg, 1994.
22. Joseph Sifakis. Property preserving homomorphisms of transition systems. In Edmund M. Clarke and Dexter Kozen, editors, *4th Workshop on Logics of Programs*, volume 164 of *Lecture Notes in Computer Science*, pages 458–473. Springer-Verlag, June 1983.
23. Bernhard Steffen, Kim G. Larsen, and Carsten Weise. A constraint oriented proof methodology based on modal transition systems. In *Tools and Algorithms for the Construction and Analysis of Systems*, volume 1019 of *Lecture Notes in Computer Science*, pages 17–40. Springer-Verlag, 1995.
24. Wolfgang Thomas. Automata on infinite objects. In Jan van Leeuwen, editor, *Handbook of Theoretical Computer Science, volume B: Formal Models and Semantics*, pages 133–194. Elsevier, Amsterdam, 1990.