In 2^{nd} *Asian Computing Science Conference, ASIAN'96*
Vol. 1179 of Lecture Notes in Computer Science, pp. 276–286
Springer-Verlag, Dec. 1996.

Hierarchical Verification using Verification Diagrams*

I.A. Browne, Z. Manna, H.B. Sipma

Computer Science Department
Stanford University, Stanford, CA 94305
anca|manna|sipma@cs.stanford.edu

1 Introduction

Graphical methods are increasingly being used in the *specification* of reactive systems, to make these specifications more readable by visualizing the intended behavior over time. However, while graphical techniques to represent *proofs* that reactive systems satisfy temporal properties have been proposed, they have had much less impact. An obvious reason for this is that (formal) specification is a much more practiced activity than formal verification. Even so, we feel that graphical representation of proofs is underutilized. In this paper we extend the formalism of verification diagrams with the objective to make them more flexible and suitable for proofs of large systems.

Verification diagrams were introduced in [MP94] as a way to represent (deductive) verification proofs. A verification diagram is a directed graph with nodes labeled by first-order assertions. A computation of such a diagram is a sequence of states that satisfy the assertions of the nodes on a walk through the diagram. Property-dependent well-formedness conditions on the graph ensure that all the computations of the diagram satisfy the property of interest. A set of first-order verification conditions are used to show that the diagram simulates the program, that is, all computations of the program are computations of the diagram. Thus, a verification diagram can be viewed as a graphical inference rule that reduces the proof of a temporal property to proofs of first-order formulas.

Although verification diagrams as defined in [MP94] form a convenient way to prove properties expressed by formulas in some canonical forms, they have some drawbacks. First, the fact that they accomodate only a restricted class of properties limits their applicability and requires often nontrivial rewriting of a formula into a canonical form. This problem was addressed in [BMS95], where the applicability of verification diagrams was extended to the entire class of quantifier-free temporal specifications. Intuitively, the diagrams were viewed as ω-automata, and the well-formedness conditions were replaced by a decidable

* This research was supported in part by the National Science Foundation under grant CCR-92-23226, the Advanced Research Projects Agency under NASA grant NAG2-892, the United States Air Force Office of Scientific Research under grant F49620-93-1-0139, the Department of the Army under grant DAAH04-95-1-0317, and a gift from Intel Corporation.

language inclusion check. A second drawback is more fundamental: verification diagrams do not scale up well. Complex properties and/or complex systems may require unmanageably large diagrams, or one has to resort to informal meta-reasoning to combine multiple diagrams.

In this paper, we propose a formal framework that allows the combination of multiple diagrams into one proof, thus facilitating incremental proof construction of complex systems and properties. In addition, we extend the applicability of verification diagrams to properties specified by existentially quantified temporal formulas, i.e. formulas of the form $\exists \overline{y}.\varphi$ where φ may contain universal quantifiers outside the scope of the temporal operators. These formulas are strictly more expressive than quantifier-free temporal formulas [Wol83].

Existentially quantified formulas are proven using augmentations of transition systems. These augmentations are obtained by adding new variables and operations on them to the system in such a way that the system and its augmentation satisfy exactly the same properties, provided these properties do not mention the new variables. An existentially quantified formula of the form $\exists \overline{y}.\varphi$ can be proven over a reactive system Φ, by constructing an augmentation $\hat{\Phi}$ of Φ which variables include $\overline{y}$, and subsequently proving φ over $\hat{\Phi}$. A proof by verification diagram may now consist of multiple diagrams, where each diagram proves a supporting property over Φ or $\hat{\Phi}$. Language intersection of the constituting diagrams is used to determine whether the collection of diagrams satisfies the property to prove.

Augmentations have been used extensively in refinement, see e.g. [AL88]. To use augmentations to prove existentially quantified temporal formulas is not new; however, it has always been done in a rather informal way.

Our method resembles that of Alpern and Schneider [AS89], but is more flexible, and, in our opinion, more intuitive. Their objective was to eliminate temporal logic and derive proof obligations directly from the specification, expressed by a collection of deterministic Büchi automata. Due to the limited expressive power of these automata a property has to be decomposed into CNF, and one automaton has to be created for each atomic formula. Verification conditions involve multiple automata, making them rather complicated. In our framework we use Streett automata, which are as expressive as existentially quantified formulas, giving us more freedom in the choice of the constituting diagrams. In addition, our verification conditions are related to individual diagrams, making them simpler.

A similar formalism to represent proofs are *Fairness Diagrams* [dAM96]. The representation in that work is geared towards a gradual construction of the proof using transformations. Another difference is that, in this formalism, verification conditions are used to relate the diagrams, making them more complex.

The remainder of this paper is organized as follows. Section 2 introduces the computational model and defines augmentations. In Section 3 we define verification diagrams. Section 4 describes how these verification diagrams can be combined into a proof. The approach is illustrated by the outline of a proof of a complex property (not expressible by a quantifier-free temporal formula).

2 Computational Model: Fair Transition Systems

The computational model we use is that of a *fair transition system* (FTS) [MP95]. An FTS is a tuple $\langle V, \Theta, \mathcal{T}, \mathcal{J}, \mathcal{C} \rangle$, where V is a finite set of variables, Θ is an initial first-order formula (assertion), $\mathcal{T}$ is a finite set of transitions, and $\mathcal{J}, \mathcal{C} \subseteq \mathcal{T}$ are the sets of just (weakly fair) and compassionate (strongly fair) transitions, respectively. Σ denotes the set of all V-states. A transition $\tau \in \mathcal{T}$ is a function $\tau : \Sigma \mapsto 2^{\Sigma}$, and each state in $\tau(s)$ is called a τ-successor of s. Each transition τ is represented by a transition relation $\rho_\tau(s, s')$, an assertion that expresses the relation between the values of V in s and the values of V (referred to by V') in any of its τ-successors s'. The enabledness of a transition τ at state s is expressed by $En(\tau)(s) : \exists s' . \rho_\tau(s, s')$.

A *run* of Φ is an infinite sequence of states such that the first state satisfies Θ and any two consecutive states satisfy a ρ_τ for some $\tau \in \mathcal{T}$. A *computation* of Φ is a run of Φ such that for each $\tau \in \mathcal{J}$ ($\tau \in \mathcal{C}$), it is not the case that τ is continually enabled beyond some point (infinitely many times enabled) but taken only finitely many times. The set of computations of Φ is denoted by $\mathcal{L}(\Phi)$.

A first-order formula φ with free variables in V is *Φ-valid*, also written $\Phi \vDash \varphi$, if it is true in all the V-states that appear in a computation of Φ.

$$
\begin{array}{lll}
\textbf{in} & N & : \textbf{integer where } N \geq 1 \\
\textbf{local} & r,\ ne,\ nf & : \textbf{integer where } r = 1,\ ne = N,\ nf = 0 \\
 & b & : \textbf{list of integer where } b = \Lambda
\end{array}
$$

$$
Prod :: \left[\begin{array}{l}
\textbf{local } x : \textbf{integer} \\
\textbf{local } t : \textbf{list} \\
\ell_0\!: \ \textbf{loop forever do} \\
\quad \left[\begin{array}{l}
\ell_1\!: \ \textbf{produce } x \\
\ell_2\!: \ \textbf{request } ne \\
\ell_3\!: \ \textbf{request } r \\
\ell_4\!: \ t := b \bullet x \\
\ell_5\!: \ b := t \\
\ell_6\!: \ \textbf{release } r \\
\ell_7\!: \ \textbf{release } nf
\end{array}\right]
\end{array}\right]
\quad \| \quad
Cons :: \left[\begin{array}{l}
\textbf{local } y : \textbf{integer} \\
m_0\!: \ \textbf{loop forever do} \\
\quad \left[\begin{array}{l}
m_1\!: \ \textbf{request } nf \\
m_2\!: \ \textbf{request } r \\
m_3\!: \ y := hd(b) \\
m_4\!: \ b := tl(b) \\
m_5\!: \ \textbf{release } r \\
m_6\!: \ \textbf{release } ne \\
m_7\!: \ \textbf{consume } y
\end{array}\right]
\end{array}\right]
$$

Fig. 1. Program PROD-CONS

Example Consider the program PROD-CONS, a shared-variable implementation of a producer-consumer protocol (Figure 1). The values produced by PROD are put in the buffer b from where they are taken by CONS. The semaphore variables r, ne and nf are used to ensure, respectively, that the two processes do not modify b at the same time, that CONS does not attempt to take a message out of an empty buffer, and that b never contains more than N pending messages. The $\bullet$ operator is used to construct new lists. The fair transition system associated with this program has one transition corresponding to each statement of the program. For instance, the transition corresponding to ℓ_1 is τ_{ℓ_1} with $\rho_{\tau_{\ell_1}} : \ell_1 \wedge \neg\ell_1' \wedge \ell_2' \wedge x' > 0 \wedge preserve(V - \{\ell_1, \ell_2, x\})$ where, for any set of variables U, $preserve(U) = \bigwedge_{v \in U}(v' = v)$. We assume that every transition system includes an *idle* transition τ_i, defined by $\rho_{\tau_i} : preserve(V)$. The idle transition is neither compassionate nor just, the transitions associated with the **request** statements are compassionate and all other transitions are just.

2.1 Augmented Transition Systems

In order to prove that an FTS Φ satisfies a property expressed by $\exists \overline{x}.\varphi$, we augment Φ with the quantified variables $\overline{x}$ such that (1) for every computation of Φ there exists a corresponding computation of the augmented system, $\hat{\Phi}$, and vice versa, and (2) the augmented system satisfies φ.

Let $\Phi : \langle V, \Theta, \mathcal{T}, \mathcal{J}, \mathcal{C}\rangle$ be an FTS. The FTS $\widehat{\Phi} : \langle \widehat{V}, \widehat{\Theta}, \widehat{\mathcal{T}}, \widehat{\mathcal{J}}, \widehat{\mathcal{C}}\rangle$ is called an augmentation of Φ if $V \subseteq \widehat{V}$ and the components of $\widehat{\Phi}$ and those of Φ have the same effect on V. More precisely, they should satisfy the following conditions

1. $\widehat{V} = V \cup U$ where $V \cap U = \emptyset$
2. $\Theta \leftrightarrow \exists U.\widehat{\Theta}$ is Φ-valid
3. $\widehat{\mathcal{T}} = \{\widehat{\tau} \mid \tau \in \mathcal{T}\}$ where $\rho_\tau(V, V') \leftrightarrow \exists U' \cdot \rho_{\widehat{\tau}}(\widehat{V}, \widehat{V}')$ is $\widehat{\Phi}$-valid
4. $\widehat{\mathcal{J}} = \{\widehat{\tau} \mid \tau \in \mathcal{J}\}$ and $\widehat{\mathcal{C}} = \{\widehat{\tau} \mid \tau \in \mathcal{C}\}$.

For any subset $W \subseteq U$, $\widehat{\Phi}$ is called a W-augmentation of Φ.

A V-state s is the V-projection of a $\widehat{V}$-state $\widehat{s}$ iff they agree on the values of the variables in V. A sequence of V-states $\sigma : s_0, s_1, \ldots$ is the V-projection of a sequence $\widehat{\sigma} : \widehat{s}_0, \widehat{s}_1, \ldots$ of $\widehat{V}$-states if for each $i \geq 0$, s_i is the V-projection of $\widehat{s}_i$.

Let Φ and $\widehat{\Phi}$ be an FTS and its augmentation. It can easily be shown that σ is a run (computation) of Φ iff there exists a run (computation) $\widehat{\sigma}$ of $\widehat{\Phi}$ such that σ is its V-projection. As a consequence, for any temporal formula φ over V, $\widehat{\Phi} \models \varphi$ iff $\Phi \models \varphi$.

3 Verification Diagrams

A verification diagram can be viewed as a finite abstraction of a reactive system. It traces the temporal behavior of the system by making explicit those parts of the behavior that are relevant to the property, and summarizing the other parts.

In general, a verification diagram Ψ that aims to prove that an FTS Φ satisfies a property φ, represents a superset of the computations of Φ, such that $\mathcal{L}(\Phi) \subseteq \mathcal{L}(\Psi) \subseteq \mathcal{L}(\varphi)$. The proof obligations spawned by a verification diagram are (1) to show that indeed all computations of Φ are represented by Ψ, and (2) to show that these computations are contained in those satisfying φ. The first part consists of proving a set of first-order verification conditions that relate the initial and consecution behavior of runs of Φ and Ψ. Conditions on ranking functions ensure that all runs of Φ are accepted by Ψ. Finally, edges may be labeled with just or compassionate transitions; these labelings may then be used to eliminate unfair runs from the language of Ψ. The second part can be done algorithmically, using standard ω-automata techniques for constructing products and checking for emptiness.

3.1 Formula Automata

Verification diagrams can be viewed as a type of ω-automata, here called formula automata. These are automata on infinite sequences of states with nodes labeled by first-order assertions and Streett-like acceptance condition on edges.

A *formula automaton* (FA) $A = \langle N, N_0, E, \mu, \mathcal{F} \rangle$ over a set of variables W has the following components:

- N: A finite set of nodes.
- $N_0 \subseteq N$: A set of initial nodes.
- $E \subseteq N \times N$: A set of edges connecting nodes.
- $\mu : N \mapsto F(W)$: A node labeling, where $F(W)$ is the set of all assertions over W.
- $\mathcal{F} \subseteq 2^E \times 2^E$: An edge acceptance list $\mathcal{F} = \{(P_1, R_1), \ldots, (P_m, R_m)\}$, where $P_j \subseteq E$ are the *persistent* edges, and $R_j \subseteq E$ are the *recurrent* edges.

An infinite sequence of nodes $\pi : n_0, n_1, \ldots$ is called a *path* of A if $n_0 \in N_0$, and for each $i \geq 0$, $(n_i, n_{i+1}) \in E$. For a path π, let $inf_e(\pi)$ be $\{e \in E \mid e$ occurs infinitely often in $\pi\}$, where we say that an edge e occurs in π if $e = (n_i, n_{i+1})$ occurs in π. A path π is *accepting* if it satisfies

$$\forall j \in [1..m] \,.\, inf_e(\pi) \subseteq P_j \;\vee\; (inf_e(\pi) \cap R_j \neq \emptyset),$$

For any infinite sequence of states $\sigma : (s_0, s_1, \ldots)$ a path of A, $\pi : (n_0, n_1, \ldots)$, is called a *trail* of σ in A if for all $i \geq 0$, s_i satisfies the labeling of n_i.

The language of an FA A, denoted by $\mathcal{L}(A)$, is the set of all infinite sequences accepted by A, also called *computations* of A, where an infinite sequence is accepted if it has an accepting trail in A. For any quantifier-free temporal property φ, there exists an FA $A(\varphi)$ whose language contains exactly those sequences that satisfy φ ([MW84]).

As mentioned before, to determine whether all computations represented by a verification diagram satisfy a property is done by a language inclusion check. To make this check decidable we define a so-called *propositional node labeling* and *propositional language* for formula automata. The propositional node labeling is then related to the first-order node labeling, by interpreting all atomic first-order formulas as propositions. More formally, let $f : N \mapsto \mathcal{P}$ be a propositional node labeling of A where $\mathcal{P}$ is a set of propositional assertions. The propositional language of A defined by f is the language of the FA $\langle N, N_0, E, f, \mathcal{F} \rangle$: $\mathcal{L}^p_f(A) \triangleq \mathcal{L}(\langle N, N_0, E, f, \mathcal{F} \rangle)$.

For a quantifier-free formula φ, let $F(\varphi)$ be the set of its first-order subformulas and let $\mathcal{P}(\varphi)$ be the set of propositional combinations of the atomic propositions in $\{P_a \mid a \in F(\varphi)\}$. We define the propositional language of φ to be $\mathcal{L}^p(\varphi) \triangleq \mathcal{L}^p_f(A(\varphi))$ where $f : N \to \mathcal{P}(\varphi)$ labels each node n by the strongest propositional assertion $f(n) \in \mathcal{P}(\varphi)$ such that $\mu(n) \to f(n)$:

$$f(n) = \bigwedge \{P_a \mid a \in F(\varphi), \mu(n) \to a\} \wedge \bigwedge \{\neg P_a \mid a \in F(\varphi), \mu(n) \to \neg a\}.$$

3.2 Definition

A *verification diagram* (VD) $\Psi = \langle N, N_0, E, \mu, \eta, \mathcal{F}, \Delta, f \rangle$ over an FTS Φ and a set of variables W has the following components:

- N, N_0, E, μ, and $\mathcal{F}$ are defined as for formula automata (Section 3.1).
- $\Delta \subseteq \{\delta \mid \delta : \Sigma \mapsto \mathcal{D}\}$: A set of ranking functions, mapping states to elements of a well-founded domain $\mathcal{D}$. For each pair (P_j, R_j) of the acceptance list, and for each node $n \in N$, Δ contains a ranking function $\delta_{j,n}$.
- $\eta : E \mapsto 2^{\mathcal{T}}$: An edge-labeling function. Each edge is labeled by a set of transitions.
- $f : N \mapsto \mathcal{P}$: A propositional node-labeling function, where $F \subseteq F(W)$ and $\mathcal{P}$ is the set of propositional combinations of the atomic propositions in $\{P_a \mid a \in F\}$.

3.3 Verification Conditions

The verification conditions associated with a VD Ψ are the following, where
$\mu(N') \triangleq \bigvee_{n \in N'} \mu(n)$,
$next(n) \triangleq \{n' \mid (n, n') \in E\}$, and
$\tau(n) \triangleq \{n' \in next(n) \mid \tau \in \eta(n, n')\}$.

Initiation : At least one initial node satisfies the initial condition of Φ:
$\Theta \rightarrow \mu(N_0)$.

Consecution : Any τ-successor of a state satisfying $\mu(n)$ satisfies the label of some node in $next(n)$:
$\forall n \in N, \forall \tau \in \mathcal{T}, \; \mu(n)(s) \wedge \rho_\tau(s, s') \rightarrow \mu(next(n))(s')$.

Acceptance : For each pair $(P_j, R_j) \in \mathcal{F}$ and for any $e = (n_1, n_2) \in E$ and $\tau \in \mathcal{T}$, taking τ will decrease (not increase) δ_j if $e \notin (P_j \cup R_j)$ (if $e \notin R_j$):

(A1) If $e \in P_j - R_j$
then $\rho_\tau(s, s') \wedge \mu(n_1)(s) \wedge \mu(n_2)(s') \rightarrow \delta_{j,n_1}(s) \succeq \delta_{j,n_2}(s')$.

(A2) If $e \in \overline{R_j \cup P_j}$
then $\rho_\tau(s, s') \wedge \mu(n_1)(s) \wedge \mu(n_2)(s') \rightarrow \delta_{j,n_1}(s) \succ \delta_{j,n_2}(s')$.

Fairness : For each $e = (n_1, n_2) \in E$ and $\tau \in \eta(e)$

(F1) τ is guaranteed to be enabled in n_1: $\mu(n_1)(s) \rightarrow En(\tau)(s)$.

(F2) A τ-successor of a state satisfying $\mu(n_1)$ satisfies the label of a node in $\tau(n_1)$: $\mu(n_1)(s) \wedge \rho_\tau(s, s') \rightarrow \mu(\tau(n_1))(s')$.

(F3) If τ is compassionate and n_3 is a node in the same connected component as n_1 and n_2 then either every loop through n_3 goes through an edge labeled τ or τ is guaranteed disabled on n_3, that is, $\mu(n_3)(s) \rightarrow \neg En(\tau)(s)$.

Satisfaction : The node-labeling μ implies the propositional node-labeling f: for all $n \in N$, $\mu(n) \rightarrow \iota(f(n))$ where $\iota : \mathcal{P} \hookrightarrow F(W)$ is the inclusion defined by $P_a \mapsto a$.

Definition 1 A verification diagram Ψ over an FTS Φ is *Φ-valid* if all the verification conditions associated with Ψ are Φ-valid.

3.4 Languages Defined by a Verification Diagram

The language of a verification diagram is the language of the corresponding formula automata whose acceptance list is augmented to include the fairness conditions associated with the edge labeling. Thus verification diagrams are as expressive as formula automata. The reason we present our formalism in terms of verification diagrams rather than formula automata is that they lead to simpler verification conditions.

Let $\Psi : \langle N, N_0, E, \mu, \eta, \mathcal{F}, \Delta, f\rangle$ be a VD. Ψ defines a formula automaton $A(\Psi) : \langle N, N_0, E, \mu, \mathcal{F}^a\rangle$ with acceptance list

$$\begin{aligned}\mathcal{F}^a = \mathcal{F}\ &\cup\ \{(P_\tau = \emptyset, R_\tau = \mathit{Taken}(\tau) \cup \mathit{Disabled}(\tau)) \mid \tau \in \mathcal{J}\}\\ &\cup\ \{(P_\tau = \mathit{Disabled}(\tau), R_\tau = \mathit{Taken}(\tau)) \mid \tau \in \mathcal{C}\}\end{aligned}$$

where $\mathit{Taken}(\tau) = \{(n_1, n_2) \in E \mid \tau \in \eta(n_1, n_2)\}$ and $\mathit{Disabled}(\tau) = \{(n_1, n_2) \in E \mid \tau(n_2) = \emptyset\}$. For each fair transition τ, the pair (P_τ, R_τ) ensures that only sequences that are fair with respect to τ are accepted.

The (first-order) language of a VD Ψ is defined by $\mathcal{L}(\Psi) \triangleq \mathcal{L}(A(\Psi))$ and the propositional language by $\mathcal{L}^p(\Psi) \triangleq \mathcal{L}^p_f(A(\Psi))$.

4 Verification Diagram Proofs

In this section we show how multiple verification diagrams can be combined in a single proof. Every verification diagram can be thought of as the proof of a lemma. Each verification diagram (lemma) is checked to be valid in the system and a final language inclusion check is used to determine that the conjunction of all the lemmas implies the property. Augmented systems are used to deal with the existentially quantified variables. In Subsection 4.2 we show that the use of previously proven properties is easy to incorporate into this formalism; it just requires to modify the language inclusion test. In Subsection 4.3 we define a set of conditions under which proving the validity of a VDP over a system can be reduced to proving its validity over a subsystem.

4.1 Definition

A verification diagrams proof, VDP for short, of a property φ over an FTS Φ is a tuple $\Upsilon : \langle \overline{x}, \overline{y}, \Psi_1, \Psi_2, \ldots, \Psi_m\rangle$, where $\overline{x}$ is a set of flexible variables, that is variables that may change value from one state to the next in a computation, $\overline{y}$ is a set of rigid variables, and $\Psi_1, \ldots, \Psi_m$ are verification diagrams over Φ and a set of variables $W \supseteq \overline{x}, \overline{y}$.

Definition 2 A VDP $\Upsilon : \langle \overline{x}, \overline{y}, \Psi_1, \ldots, \Psi_m\rangle$ over an FTS Φ is *Φ-valid* if there exists an $\overline{x}$-augmentation $\widehat{\Phi}$ of Φ such that Ψ_i is $\widehat{\Phi}$-valid for each $i \in [1..m]$.

An infinite sequence σ of states is a *computation* of Υ if it is a computation of each Ψ_i, $i \in [1..m]$. The *language* defined by Ψ, $\mathcal{L}(\Psi)$, is the set of its computations. Thus

$$\mathcal{L}(\langle \overline{x}, \overline{y}, \Psi_1, \ldots, \Psi_m \rangle) \triangleq \bigcap_{i=1}^{m} \mathcal{L}(\Psi_i) \quad \text{and} \quad \mathcal{L}^p(\langle \overline{x}, \overline{y}, \Psi_1, \ldots, \Psi_m \rangle) \triangleq \bigcap_{i=1}^{m} \mathcal{L}^p(\Psi_i).$$

Definition 3 Let Φ be an FTS with variables V. An $\exists$-*formula* is a temporal formula with no free variables of the form $\exists\overline{x}\forall\overline{y}.\varphi$, where φ is quantifier-free and $\overline{x}$ ($\overline{y}$) are flexible (rigid) variables not in V.

Definition 4 Let Φ be a fair transition system and $\varphi : \exists\, \overline{x}^e \forall \overline{y} \cdot \varphi^e$ an $\exists$-formula. A VDP $\Upsilon : \langle \overline{x}, \overline{y}, \Psi_1, \ldots, \Psi_m \rangle$ over Φ proves that Φ satisfies φ, written $\Phi \overset{\Upsilon}{\models} \varphi$, if (1) $\overline{x}^e \subseteq \overline{x}$, (2) Υ is Φ-valid, and (3) $\mathcal{L}^p(\Upsilon) \subseteq \mathcal{L}^p(\varphi^e)$.

Theorem 1. (Soundness and Completeness) *Given an* FTS Φ *and an* $\exists$-*formula,* $\varphi = \exists\overline{x}\forall\overline{y} \cdot \varphi^e$, $\Phi \models \varphi$ *iff there exists an* VDP Υ *such that* $\Phi \overset{\Upsilon}{\models} \varphi$.

4.2 Use of Previously Proven Properties

Previously proven (temporal) properties can be used to eliminate computations from a verification diagram. This facility often leads to smaller diagrams since the details of the parts of the computations described by these previously proven properties do not need to be represented in the diagram.

Definition 5 Let Φ be an FTS, $\alpha_1, \ldots, \alpha_k$ be quantifier-free temporal formulas and $\varphi : \exists\, \overline{x}^e \forall \overline{y} \cdot \varphi^e$ be an $\exists$-formula. We say that the VDP $\Upsilon :\ \langle \overline{x}, \overline{y}, \Psi_1, \ldots, \Psi_m \rangle$ proves that Φ satisfies φ using $\alpha_1, \ldots, \alpha_k$, written $\alpha_1, \ldots, \alpha_k, \Phi \overset{\Upsilon}{\models} \varphi$, if (1) $\overline{x}^e \subseteq \overline{x}$, (2) Υ is Φ-valid, and (3) $\mathcal{L}^p(\Upsilon) \cap \mathcal{L}^p(\alpha_1) \cap \ldots \cap \mathcal{L}^p(\alpha_k)) \subseteq \mathcal{L}^p(\varphi^e)$.

With this definition we can formulate rule VDP-H shown in Figure 2, which allows inclusion of previously proven properties in a proof by verification diagram.

For an FTS Φ, an $\exists$-formula φ, a VDP Υ and quantifier-free formulas $\alpha_1, \ldots, \alpha_k$

$$\frac{\begin{array}{c}\Phi \models \alpha_1, \ \ldots \ \Phi \models \alpha_k \\ \alpha_1, \ldots, \alpha_k, \Phi \overset{\Upsilon}{\models} \varphi\end{array}}{\Phi \models \varphi}$$

Fig. 2. Rule VDP-H

When using an augmented system $\widehat{\Phi}$ to prove an existentially quantified formula, it is often convinient to use verification diagrams or formulas that were proven valid in the original system Φ. In Subsection 2.1 we mentioned that a formula over V holds on Φ iff it holds on $\widehat{\Phi}$. The analogous result is true for verification diagrams.

Lemma 2. *If $\widehat{\Phi}$ is an augmentation of an* FTS *Φ and Υ is a* VDP *over Φ and a set of variables W such that $W \cap (\widehat{V} - V) = \emptyset$ then Υ is Φ-valid iff Υ is $\widehat{\Phi}$-valid.*

4.3 Reduction to Subsystems

Often a property of a system is dependent only on a subset of the transitions of that system. In this case it is advantageous to be able to prove the property valid for this subsystem and then infer its validity over the entire system. The following definition and lemma define a set of conditions under which this is possible.

Definition 6 Let Φ be an FTS, Υ a VDP over Φ and a set of variables W, and $U \subseteq W$. We say that Υ *admits any U-modifications* in Φ if for any $(W-U)$-state s_1, U-states s_2, s_2', and Φ-reachable states (s_1, s_2) and (s_1, s_2'), the following hold

- Any path through the verification diagram can always be extended when variables in U are modified, that is, for any $n \in N$,

$$\mu(n)(s_1, s_2) \;\rightarrow\; \mu(next(n))(s_1, s_2')$$

- Any change to the of variables in U must not violate the *Acceptance* condition of the diagram. More precisely, for any $(P_j, R_j) \in \mathcal{F}$,

$$\begin{array}{l} \text{if } (n_1, n_2) \in P_j - R_j \\ \text{then } \mu(n_1)(s_1, s_2) \wedge \mu(n_2)(s_1, s_2') \;\rightarrow\; \delta_{j,n_1}(s_1, s_2) \succeq \delta_{j,n_2}(s_1, s_2') \\ \text{if } (n_1, n_2) \in \overline{P_j \cup R_j} \\ \text{then } \mu(n_1)(s_1, s_2) \wedge \mu(n_2)(s_1, s_2') \;\rightarrow\; \delta_{j,n_1}(s_1, s_2) \succ \delta_{j,n_2}(s_1, s_2') \end{array}$$

Based on this definition, the following lemma infers the $\Phi' \parallel \Phi''$-validity of a VDP Υ from its validity over the subsystem Φ'.

Lemma 3. *Let Φ' and Φ'' be two* FTS*s, Ψ a verification diagram and W'' a set of variables that includes all the variables modified by Φ'' that are not in V'. If Υ is Φ'-valid and Υ admits any W''-modifications in $\Phi' \parallel \Phi''$ then Υ is $\Phi' \parallel \Phi''$-valid.*

4.4 Example

We illustrate the techniques described in this paper by proving that in all PROD-CONS computations the values are consumed in the order in which they are produced (φ_3) and that the producer (consumer) will keep producing (consuming) (φ_4 and φ_5). This property is expressible by the following $\exists$-formula in which the *history variables* buf_p and buf_c record the values produced (φ_1) and consumed (φ_2) respectively.

φ_{order} : $\exists buf_p, buf_c \forall u, v, np > 0, nc > 0 \cdot (\varphi_1 \wedge \varphi_2 \wedge \varphi_3 \wedge \varphi_4 \wedge \varphi_5)$ where

$$\begin{array}{ll}
\varphi_1 : buf_p = \Lambda \;\wedge\; \Box(buf_p = u \rightarrow & \mathit{if}\ (\ell_1 \wedge \bigcirc \neg \ell_1)\ \mathit{then}\ \bigcirc(buf_p = u \bullet x) \\
& \mathit{else}\ \ \bigcirc(buf_p = u)) \\
\varphi_2 : buf_c = \Lambda \;\wedge\; \Box(buf_c = v \rightarrow & \mathit{if}\ (m_7 \wedge \bigcirc \neg m_7)\ \mathit{then}\ \bigcirc(buf_c = v \bullet y) \\
& \mathit{else}\ \ \bigcirc(buf_c = v))
\end{array}$$

$\varphi_3 : \Box(buf_c \subseteq buf_p)$ $\qquad$ $\varphi_4 : \Diamond(|buf_p| > np)$ $\qquad$ $\varphi_5 : \Diamond(|buf_c| > nc)$

We first define $\widehat{\text{PROD-CONS}}$, a $\{buf_p, buf_c\}$-augmentation of PROD-CONS.

$$\begin{array}{llll}
\widehat{\Theta} & = \Theta \wedge buf_p = \Lambda \wedge buf_c = \Lambda & & \\
\widehat{\rho}_{\tau_{\ell_1}} & = \rho_{\tau_{\ell_1}} \wedge buf'_p = buf_p \bullet x & \widehat{\rho}_{\tau_{m_7}} = \rho_{\tau_{m_7}} \wedge buf'_c = buf_c \bullet y & \\
\widehat{\rho}_{\tau} & = \rho_{\tau} \wedge \mathit{preserve}\{buf_p, buf_c\} & \text{for } \tau \notin \{\tau_{\ell_1}, \tau_{m_7}\} &
\end{array}$$

Next, we show that it satisfies all the quantifier-free conjuncts φ_1-φ_5, by constructing a separate VD, $\Psi_1 - \Psi_5$, for each of these formulas. The verification diagram Ψ_1 is shown in Figure 3. Encapsulation conventions are used to make the diagram more concise [Har87], and selfloops on all the basic nodes are omitted. Ψ_2 can be constructed in a similar way. The invariant $buf_c \subseteq buf_p$ can be proven using a VD Ψ_2 with a single node labeled $buf_p = buf_c \bullet (\mathit{if}\ m_{5..7}\ \mathit{then}\ y\ \mathit{else}\ \Lambda) \bullet b \bullet (\mathit{if}\ \ell_{2..5}\ \mathit{then}\ x\ \mathit{else}\ \Lambda)$.

In the proof of φ_4 and φ_5 we make use of the verification rule VDP-H and Lemma 3. It is easy to check that the verification diagram Ψ_4, shown in Figure 4, is $\widehat{\text{PROD}}$-valid. As Ψ_4 does not refer to any variables modified by $\widehat{\text{CONS}}$, it is, by Lemma 3, also $\widehat{\text{PROD-CONS}}$-valid. The diagram allows a computation to either stay in n_0 or n_1 forever, or eventually reach n_2, and thus satisfy $|buf_p > np|$.

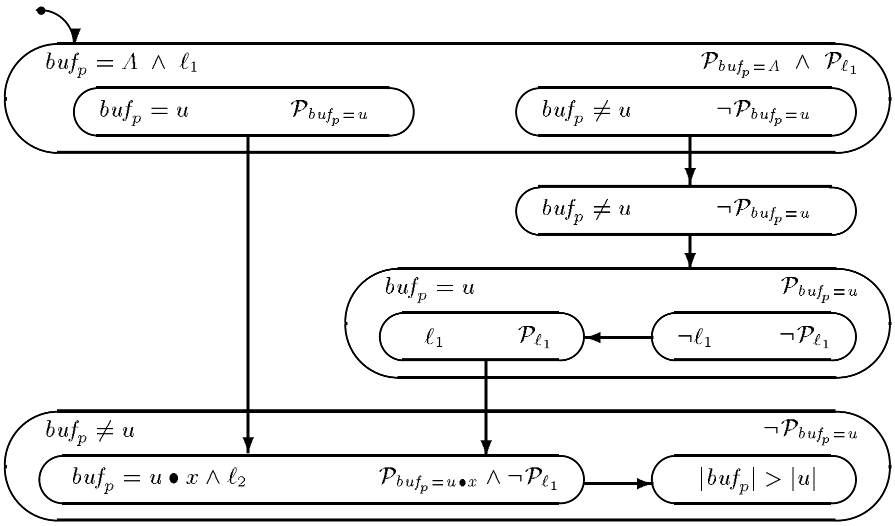

Fig. 3. Verification diagram Ψ_1: $\mathcal{F} = \emptyset$, $\Delta = 0$

It is straightforward to show that PROD-CONS satisfies $\Box\Diamond\ell_1$, using the fact that it satisfies $\Box\Diamond(r > 0)$ and $\Box\Diamond(ne > 0)$, which can be readily shown by a verification diagram. Thus also $\widehat{\text{PROD-CONS}}$ satisfies $\Box\Diamond\ell_1$. This property eliminates all computations that forever stay in n_0 or in n_1, and so we have $\Box\Diamond\ell_1, \widehat{\text{PROD-CONS}} \overset{\Psi_4}{\models} \Diamond(|buf_p| > np)$. In a similar way we can construct a VD Ψ_5 such that $\Box\Diamond m_7, \widehat{\text{PROD-CONS}} \overset{\Psi_5}{\models} \varphi_5$. Let $\Upsilon = \langle\{buf_p, buf_c\}, \{u, v, np > 0, nc > 0\}, \Psi_1, \Psi_2, \Psi_3, \Psi_4, \Psi_5\rangle$. We showed that $\Box\Diamond\ell_1, \Box\Diamond m_7, \text{PROD-CONS} \overset{\Upsilon}{\models} \varphi_{order}$ and, having proven $\Box\Diamond\ell_1$ and $\Box\Diamond m_7$, we can conclude, by rule VDP-H, that PROD-CONS satisfies φ_{order}.

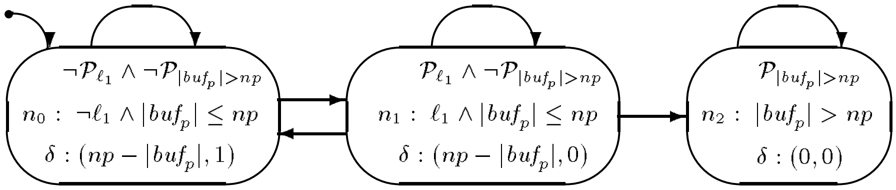

Fig. 4. Verification diagram Ψ_4 : $\mathcal{F} = \{(P : \{(n_0, n_0), (n_1, n_1)\}, R : \{(n_2, n_2)\})\}$

Acknowledgements: We thank T. Uribe and L. de Alfaro for their comments.

References

[AL88] M. Abadi and L. Lamport. The existence of refinement mappings. In *Proc. 3rd IEEE Symp. Logic in Comp. Sci.*, pages 165–175. IEEE Computer Society Press, 1988.

[AS89] B. Alpern and F.B. Schneider. Verifying temporal properties without temporal logic. *ACM Trans. Prog. Lang. Sys.*, 11:147–167, 1989.

[BMS95] A. Browne, Z. Manna, and H.B. Sipma. Generalized verification diagrams. In *15th Conference on the Foundations of Software Technology and Theoretical Computer Science*, volume 1026 of *LNCS*, pages 484–498, 1995.

[dAM96] L. de Alfaro and Z. Manna. Temporal verification by diagram transformations. In *Proc. 8^{th} Intl. Conference on Computer Aided Verification*, volume 1102 of *LNCS*, pages 287–299, July 1996.

[Har87] D. Harel. Statecharts: A visual formalism for complex systems. *Sci. Comp. Prog.*, 8:231–274, 1987.

[MP94] Z. Manna and A. Pnueli. Temporal verification diagrams. In *Proc. Int. Symp. on Theoretical Aspects of Computer Software*, volume 789 of *LNCS*, 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(1):68–93, January 1984.

[Wol83] P. Wolper. Temporal logic can be more expressive. *Information and Control*, 56:72–99, 1983.

This article was processed using the $\LaTeX$ macro package with LLNCS style