Bridging System Views *

Max Breitling and Jan Philipps

Institut für Informatik
Technische Universität München
80290 München, Germany
{max.breitling|jan.philipps}@in.tum.de

Abstract. System specification by state machines together with property specification and verification by temporal logics are by now standard techniques to reason about the control flow of hardware components and embedded systems. The techniques to reason about the dataflow within loosely coupled systems, however, are less well developed.
In this contribution, we propose a formalism for the verification of systems with asynchronously communicating components. The components themselves are specified as state machines, while the dataflow between components is described as a relation over the input and output histories of a system. Communication history properties are derived from temporal logic properties of the component state machines.
For temporal reasoning we employ verification diagrams and theorem provers. We sketch the use of our approach with two examples, a communication system and the NetBill protocol.

1 Introduction

State transition diagrams in various incarnations have become a popular technique to specify software and hardware systems. Their suggestive notation leads to readable design documents for a component's implementation. With temporal logics there are precise property specification and verification techniques for state machines. Temporal logic proofs often follow the operational intuition behind state machines: Invariants, for example, are typically shown using induction over the machine transitions.

For systems with loosely coupled components, which communicate asynchronously via communication channels, however, temporal logics and model checking are less successful. Here a black box view which just relates input and output is more useful than the state-based glass box view of a component. Black box properties of dataflow components and systems can be concisely formulated as relations over the communication history of components [8]; such properties are inherently modular and allow easy reasoning about the global system behavior.

State based and history based descriptions of systems and components can be integrated [1, 3, 5, 7]. History properties of a component are derived from

* This work has been partially funded by the Sonderforschungsbereich 342 of the DFG.

invariance and response properties in temporal logic together with a continuity argument. For a high-level view on the temporal logic proofs we adopt Manna and Pnueli's verification diagrams [14]; tool support is provided through the theorem prover Isabelle [11, 17].

The utility of this approach is that is allows formulation of system properties at different abstraction levels and compositional reasoning about dataflow properties within a unified mathematical framework; we shortly describe two case studies that highlight these points.

2 Distributed Systems

We model a system by describing its interface to the system's environment, its components and the component interconnection structure. The components are connected via directed channels. The system's *interface* is described by the communication channels with the types of the message that are sent on them. The communication along all channels is modeled by finite or infinite message streams. Component *behavior* is described by state machines; the behavior of the complete system is defined by the behavior of its components.

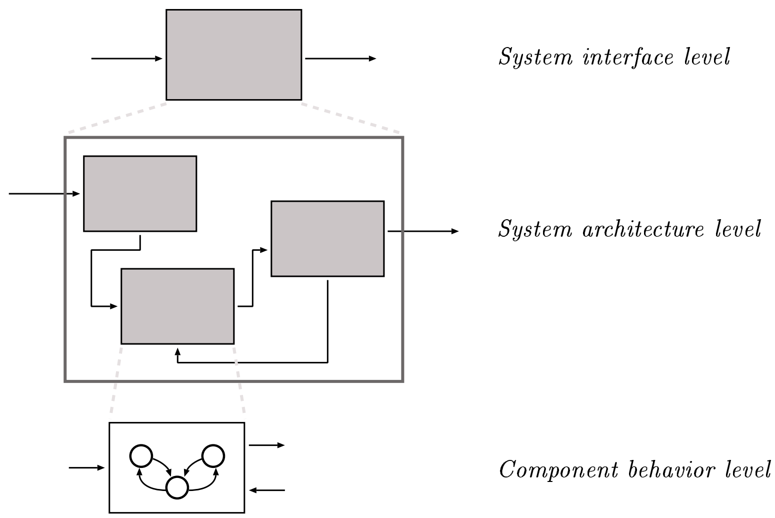

Fig. 1. Distributed System Levels

This system model leads to various abstraction levels in the design of distributed systems (Fig. 1). The most abstract level regards the whole system as a black box, and only refers to the interfaces between system and environment. The next level also refers to the internal architecture of the system; it can be regarded as a hierarchical decomposition of the interface level. In the third level, the internal behavior of the components, modeled by state transition systems, becomes visible. For some concrete state transition system notations — like Statecharts or ROOMcharts — hierarchical decomposition could be applied to this level, too.

In the rest of this section, we give an overview over the formalization of these levels. The black box aspects of the formalization are taken from the FOCUS system model [8]; the formalization of the state transition systems is elaborated in [1].

2.1 Streams

The communication history between components is modeled by *streams*. A stream is a finite or infinite sequences of messages. Finite streams can be enumerated, for example: $\langle 1, 2, 3, \ldots 10 \rangle$; the empty stream is denoted by $\langle\,\rangle$. For a set of messages M, the set of finite streams over M is denoted by M^*, that of infinite streams by M^∞. By M^ω we denote $M^* \cup M^\infty$.

Given two streams s, t and $j \in \mathbb{N}$, $\#s$ denotes the length of s. If s is finite, $\#s$ is the number of elements in s; if s is infinite, $\#s = \infty$. We write $s \frown t$ for the concatenation of s and t. If s is infinite, $s \frown t = s$. We write $s \sqsubseteq t$, if s is a prefix of t, i.e. if $\exists\, u \in M^\omega \bullet s \frown u = t$. The j-th element of s is denoted by $s.j$, if $1 \leq j \leq \#s$; it is undefined otherwise. $\mathsf{ft}.s$ denotes the first element of a stream, i.e. $\mathsf{ft}.s = s.1$, if $s \neq \langle\,\rangle$. For $A \subseteq M$ we denote by $A \circledS s$ the subsequence that results from s by removing all elements not in A.

2.2 State Transition View

A state transition system (STS) is a tuple $\mathcal{S} = (I, O, A, \mathcal{I}, \mathcal{T})$, where I, O, A are sets of variables. A state of our system is described by a valuation that assigns values to these variables. The variables in I and O represent input and output channel histories, respectively. They range over finite message streams. The set A contains variables to store local state attributes as e.g. the control state. It is assumed that for every $i \in I$ we have i° in A describing the part of the input that is already consumed. $\mathcal{I}$ is an assertion that characterizes the initial states of the state transition system. $\mathcal{T}$ is a finite set of transitions; each transition $\tau \in \mathcal{T}$ is an assertion that relates a current state with its successor states. The free variables of $\mathcal{I}$ range over $V = I \cup O \cup A$. The free variables of each transition assertion τ belong to $V \cup V'$, where $V' = \{\ v' \mid v \in V\ \}$ results from V by priming each variable. Intuitively, the unprimed variables in a transition assertion will refer to the current state, the primed variables to the subsequent state. Each transition τ can only extend the channel histories I and O. In addition, there is an environment transition τ_ϵ that leaves all variables unchanged except those of I — the input histories I may be extended.

Priming is extended to assertions and to variable valuations: Φ' is the assertion obtained from Φ by priming each free variable; from a valuation σ we get σ' via $\sigma'(v') \stackrel{\text{df}}{=} \sigma(v)$.

Since states are variable valuations for the system variables, we can talk about the validity of assertions in a state or in a pair of states. Given two assertions Φ and Ψ with free variables from V and $V \cup V'$, respectively, we write

$$\sigma \models \Phi \quad \text{and} \quad \sigma, \rho' \models \Psi$$

if Φ is true when all free variables are replaced by their values from σ, and Ψ is true when all unprimed free variables are replaced by the values from σ and all primed free variables are replaced by their values from ρ'.

An STS can be graphically represented by a *state transition diagram.* STDs are directed graphs where the vertices represent (control) states and the edges represent transitions between states. One vertex is a designated *initial state*; graphically we mark this vertex by an opaque circle in its left half. Edges are labeled; each label consists of four parts: A *precondition*, a set of *input statements*, a set of *output statements* and a *postcondition.* In STDs, transition labels are represented with the following schema:

$$\{\text{Precondition}\}\ \text{Inputs} \triangleright \text{Outputs}\ \{\text{Postcondition}\}$$

where *Inputs* and *Outputs* denote lists of expressions of the form $i?x$ and $o!exp$, respectively, with $i \in I$, $o \in O$, x a constant value or a (transition-local) variable of the type of i, and *exp* is an expression of the type of o. The *Precondition* is a boolean formula containing data state variables and transition-local variables as free variables, while *Postcondition* and *exp* may additionally contain primed state variables. The distinction between pre- and postconditions does not increase the expressiveness, but improves readability. If the pre- or postconditions are tautologies, they can be omitted.

The informal meaning of a transition is as follows: If the available messages in the input channels can be matched with *Inputs*, the precondition holds, and the postcondition can be made true by assigning proper values to the primed variables, the transition is enabled. If it is chosen, the inputs are read, the outputs are written and the postcondition is made true.

As a rather simple example, we specify a buffer by the STD in Fig. 2. Its initial state is *Empty.* If some data is received on i, it is stored in q, and the control moves to the state *Store* (according transition τ_1). In this state, receiving a request message Ⓡ, the first element of q is sent. Depending on the length of q, the buffer leaves the control state unchanged or moves back to *Empty* (τ_2 or τ_4). Receiving further data in the state *Store*, they are appended in q (τ_3). If there are no stored messages (in the state *Empty*), but a request arrives, this open request has to be remembered by incrementing c (τ_5 and τ_7). If $c > 0$, the buffer is in the state *Count.* If some data arrives now, it is immediately forwarded on o, decrementing c, until there are no more pending requests and the buffer returns to the state *Empty* (τ_6 resp. τ_8)

The state transition defines the abstract syntax of state machines; their semantics is a set of executions. An *execution* ξ is an infinite stream of valuations for the system variables that satisfies the following three requirements:

1. The first valuation in ξ satisfies the initialization assertion: $\xi.1 \models \mathcal{I}$

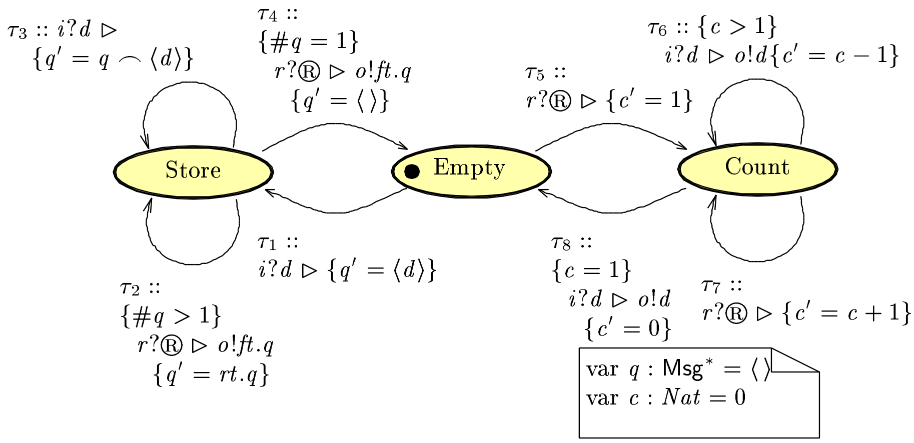

Fig. 2. State Transition Diagram for the Buffer

2. Each pair of subsequent valuations $\xi.k$ and $\xi.(k+1)$ are related either by a system or by the environment transition: $\xi.k, \xi.(k+1)' \models \tau_\epsilon \vee \bigvee_{\tau \in \mathcal{T}} \tau s$. It is because of the environment transitions that we have infinite executions even for finite input histories.
3. Each transition of the STS is taken infinitely often in an execution, unless it is disabled infinitely often (weak fairness). Of course, other fairness assumptions can also be used.

The set of executions of a state transition system $\mathcal{S}$ is denoted by $\langle\!\langle \mathcal{S} \rangle\!\rangle$.

On top of the executions, a simple linear temporal logic can be defined in the usual way[12, 13]. A predicate Φ (written as $\Box\Phi$) is an *invariant* for a system $\mathcal{S}$ iff Φ is valid in all states in all executions of $\mathcal{S}$:

$$\mathcal{S} \models \Box\Phi \quad \text{iff} \quad \forall \xi \in \langle\!\langle \mathcal{S} \rangle\!\rangle \bullet \forall k \in \mathbb{N} \bullet \xi.k \models \Phi$$

A *response* property is written as $\Phi \mapsto \Psi$. It states that whenever in an execution of the state machine $\mathcal{S}$ a state is reached where Φ holds, then at the same or a later state in the execution Ψ also holds:

$$\mathcal{S} \models \Phi \mapsto \Psi \quad \text{iff} \quad \forall \xi \in \langle\!\langle \mathcal{S} \rangle\!\rangle \bullet (\forall k \in \mathbb{N} \bullet \xi.k \models \Phi \Rightarrow \exists\, l \geq k \bullet \xi.l \models \Psi)$$

2.3 Black Box Specification

A history or black box specification is an abstract description in the sense that it does not refer to the internal state of a component, but just describes the external, visible behavior.

The behavior relation is defined by formulas Φ where the free variables range over the component input and output stream variables. The streams fulfilling these predicates describe the allowed black box behavior of our system. We can use all the operators on streams from Section 2.1 to formulate the predicates.

As a rather simple example, the buffer component can be specified abstractly by the following predicate:

$$o \sqsubseteq i \wedge \#o \leq \#r \wedge \#o \geq \mathit{min}(\#i, \#r)$$

The first conjunct states that only data received on channel i may be output on channel o. The second conjunct demands that the component never outputs more data than has been requested on channel r. In contrast with these two conjuncts, which are safety properties, the third conjunct is a liveness property: It demands that output is indeed produced.

From component history specifications it is straightforward to reach the history specification of the complete system. At the black box level, composition is just the logical conjunction of the component history specifications; internal channels can be hidden by existential quantification.

3 System Views

The different abstraction levels in a system description can be classified as state based views and as history based views. In this section, we relate the two views.

Within a state machine execution $\xi \in \langle\!\langle \mathcal{S} \rangle\!\rangle$, changes in the valuations for the input and output variables I, O are restricted to the prefix order $\sqsubseteq$: For each variable $v \in I \cup O$ and every $k \in \mathbb{N}$,

$$(\xi.k)(v) \sqsubseteq (\xi.(k+1))(v)$$

Thus the valuations of each input and output variable within an execution form a chain under the prefix order, and for each execution and each variable $v \in I \cup O$ there is a least upper bound

$$\xi.\infty(v) \stackrel{\text{df}}{=} \bigsqcup\{\ (\xi.k)(v)\ \mid\ k \in \mathbb{N}\ \}$$

Note that $\xi.\infty(v)$ is only defined for the input and output variables, not for the attribute variables of a state machine.

Based on the least upper bounds of the channel history valuations, we can define the *black box view* of a state machine $\mathcal{S} = (I, O, A, \mathcal{I}, \mathcal{T})$ as a set of valuations for the variables $I \cup O$. It is denoted by $[\![\mathcal{S}]\!]$ and defined via the least upper bounds of the input and output histories of the machine's executions:

$$[\![\mathcal{S}]\!] = \{\ \alpha \in (I \cup O) \rightarrow M^{\omega}\ \mid$$
$$\exists\, \xi \in \langle\!\langle \mathcal{S} \rangle\!\rangle \bullet \bigwedge_{i \in I} \alpha(i) = \xi.\infty(i) \wedge \bigwedge_{o \in O} \alpha(o) = \xi.\infty(o)\ \}$$

The definition shows that the "missing link" between the state machine view and the black box view of a component is the set of executions of the state machine. This motivates an additional view of components, the *execution view*.

As shown in Fig. 3, for each of the three levels there is a corresponding property specification language. These are, with rising abstraction level:

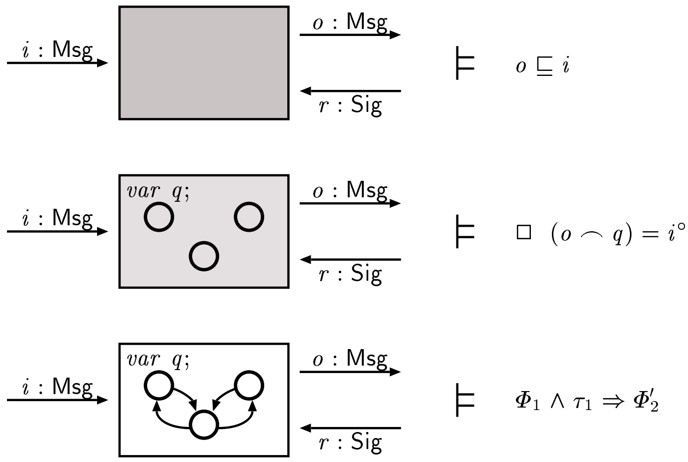

Fig. 3. Black Box, Execution and State Transition Views

- The *state transition view* describes the component with the abstract syntax of state transition systems. Properties at this level are expressed in predicate logic. Typical properties are consequence properties, which describe the effect a given transition has (e.g., $\Phi \wedge \tau \Rightarrow \Psi'$), or enabledness properties, which state that under certain conditions a given transition is *enabled* (e.g., $\Phi \Rightarrow \mathsf{En}(\tau)$, where $\sigma \models \mathsf{En}(\tau)$ iff there is a state ρ such that $\sigma, \rho' \models \tau$).
- The *execution view* describes the component by the set of its executions. Properties at this level can only refer to states that are reachable in an execution, or to the causal relation between states in an execution. The property language here is linear temporal logic. Typical properties are invariance and response formulas as defined in Section 2.2.

  Note that in the execution view, the transitions are hidden: Properties can only refer to the component state space, including the input and output channel variable valuations.
- The *black box view* describes the component by its the relation of its input and output communication histories. Properties at this level are expressed as formulas over streams (Section 2.1). Typical properties state that a stream is a substream or a prefix of another stream, or relate the length of two or more streams.

  In the black box view, both transitions and the state space of the component are hidden. Properties can only refer to channel variable valuations.

Unfortunately, the above definition of the black box semantics of a state transition system is difficult to use in practice. In the rest of this section, we introduce proof rules to derive execution view properties from state transition properties, and black box properties from execution view properties.

3.1 From Transitions to Executions

Temporal logical properties of the executions of a state machine can be derived by the usual proof rules. As examples, we state the following two:

$$\begin{array}{|l} \mathcal{I} \Rightarrow \Phi \\ \forall \tau \bullet \;\; \Phi \wedge \tau \Rightarrow \Phi' \\ \hline \mathcal{S} \models \Box \Phi \end{array} \qquad\qquad \begin{array}{|l} \forall \tau \bullet \Phi \wedge \tau \Rightarrow \Phi' \vee \Psi' \\ \exists \tau \bullet \Phi \Rightarrow \mathsf{En}(\tau) \\ \qquad \Phi \wedge \tau \Rightarrow \Psi' \\ \hline \mathcal{S} \models \Box(\Phi \Rightarrow \Diamond \Psi) \end{array}$$

A state property Φ is an invariant if it is valid initially and stays valid for all executed transitions. The response property can be shown by proving that there is a helpful transition that – once Φ is valid in a state – is enabled and leads to a successor state with Ψ valid. Due to the assumed fairness this transition will be selected eventually in an execution.

System properties formulated by temporal logic no longer contain any reference to the transitions, they just make a statement about the properties of exections being sequences of states of a system.

3.2 From Executions to Histories

The black box view of a system focuses on the external behaviour, while the execution view also contains references to internal values of the system. To derive a black box view of a system from its properties of its executions we extract the value of the input and output streams of a system in the virtual "last" state of an execution.

A system invariant Ψ stays valid in all states during an execution. If it is also *admissible* [16], Ψ is valid for an infinite execution, as it is expressed in the following rule, suitable to show safety history properties of a system.

$$\begin{array}{|l} \mathsf{adm}\ \Phi \\ \mathcal{S} \models \Box \Phi \\ \hline [\![\mathcal{S}]\!] \Rightarrow \Phi \end{array} \qquad\qquad \begin{array}{|l} \mathcal{S} \models \#o = \nu \wedge \nu < \ell \;\mapsto\; \#o > \nu \\ \hline [\![\mathcal{S}]\!] \Rightarrow \#o \geq \ell \end{array}$$

The second rule allows the derivation of a specific liveness property that we call *output extension.* If we can prove that a system eventually extends an output stream as long as a certain value ℓ is not reached, we can conclude that the black box behavior of the system assures that the length of the output will reach at least the length ℓ. The free variables of the $\mathbb{N}$-valued expression ℓ must belong to $I \cup O$.

With these two rules a major class of black box properties can be derived from the execution view of a system. While invariance is suitable to express the functional behavior, output extension enables the derivation of liveness properties. Proofs of both rules can be found in [1].

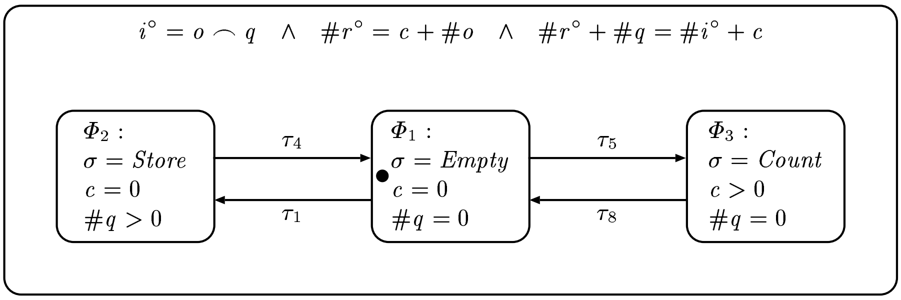

Fig. 4. Invariance Diagram for the Buffer

4 Proof Support

For practical proving we describe how verification diagrams can be used in our setting. Finally, we sketch how the vast number of proof obligations can be shown almost automatically by using tool support.

4.1 Verification Diagrams

Since invariance and response properties link the state machines and black box views of a system, proofs can be reduced to proofs of temporal logic properties. The usual linear presentation of such proofs, however, does not reflect the operational intuition behind the proof and can be confusing and hard to understand.

Verification diagrams [14] visualize the proof structure of temporal logic proofs and reduce temporal reasoning to proofs of verification conditions in first-order predicate logic. Verification diagrams can be tailored to the invariance and response properties used in the derivation of black box properties for state machines.

A verification diagram is a directed graph without unreachable nodes. The diagram's nodes are labeled by assertions $\Phi_0, \ldots, \Phi_n$. Nodes marked by opaque circles in the left half are called *initial nodes*. A node marked by an opaque circle in the right half is called the *terminal node*. Initial and terminal nodes are optional, and there must be at most one terminal node. We tacitly assume that all node assertions are syntactically different and logically exclusive, and refer to the nodes by just their assertions. The edges in a verification diagram are labeled by transitions $\tau \in \mathcal{T}$.

Verification diagrams can be hierarchical: A node can contain a sub-diagram. Hierarchical diagrams are equivalent to flattened diagrams, where assertions from a node higher in the hierarchy are conjoined with the assertion of the nodes below it and arrows entering or exiting higher-level nodes are connected to all lower-level nodes.

Invariance Diagrams. An invariance diagram is a verification diagram without a terminal node. Figure 4 shows an invariance diagram for the buffer. It is used to prove that the following formula is an invariant of the buffer, i.e.

$$Buffer \models \Box\ (\ i^\circ = o \frown q \quad \wedge \quad \#r^\circ = c + \#o \quad \wedge \quad \#r^\circ + \#q = \#i^\circ + c\ )$$

To find such an invariant, an understanding of the operation of the STS is necessary. For our example, the intended meaning of the variables q and c must be encoded in the formulas: Messages read from i are either already output on o, or are still stored in q. Received requests are either still pending (counted in c) or are already answered (by sending a message on o). The difference $\#r^\circ - c$ of the number of received requests and the number of open requests is the number of answered requests, and therefore equal to the number of received messages ($\#i^\circ$) minus the number of messages still buffered ($\#q$). Thus, the *Buffer* can have messages in q, or it can have pending requests, or can be in a balanced state where q is empty and there are no pending requests. These three case are reflected in the three nodes Ψ_1, Ψ_2 and Ψ_3 of our diagram.

All in all, there are 27 proof obligations associated with the diagram that need to be proved to show that the diagram is valid. For each node and each of the nine transitions τ_j (including the environment transition) we have to show

$$\Psi_i \wedge \tau_j \Rightarrow \Psi_k'$$

with Ψ_i denoting the source node and Ψ_k describing the target node. If τ_j maintains Ψ_i, i.e. $i = k$, we do not represent the associated loop in the diagram. If a transition τ_j is not enabled in a node Ψ_i, the left-hand side conjunct includes a contradiction and the implication is trivially true. In our example, only 11 obligations are non-trivial: Three obligations for the environment transitions, four obligations for the arrows in Fig. 4 and four obligations corresponding to transitions $\tau_2, \tau_3, \tau_6, \tau_7$ in Fig. 2.

Response Diagrams. A response diagram is a verification diagram that is acyclic: Its nodes can be ordered such that for each pair of nodes Φ_i and Φ_j, if there is an edge from Φ_i to Φ_j, then $i > j$. There is a single node with no outgoing edges. This node is marked as the terminal node and labeled with the assertion Φ_0.

Figure 5 shows a response diagram for our buffer. It is used to show the following property, which states that the buffer outputs a message on channel o provided there are enough message inputs and requests:

$$Buffer \models \#o = k \wedge k < min(\#i, \#r) \mapsto \#o > k$$

This property holds immediately for states where only transitions are enabled that produce output (τ_2, τ_4, τ_6 and τ_8). From all other states, the system must move closer to a state where output must be produced. In the verification diagram, the state space is split into five partitions, Φ_1 to Φ_5. The terminal node Φ_0 is the target node, where output on o has been produced. Transitions that send a message on o immediately reach the target node. Other transitions may keep a node assertion valid, or lead to a node closer to the target. Proofs of

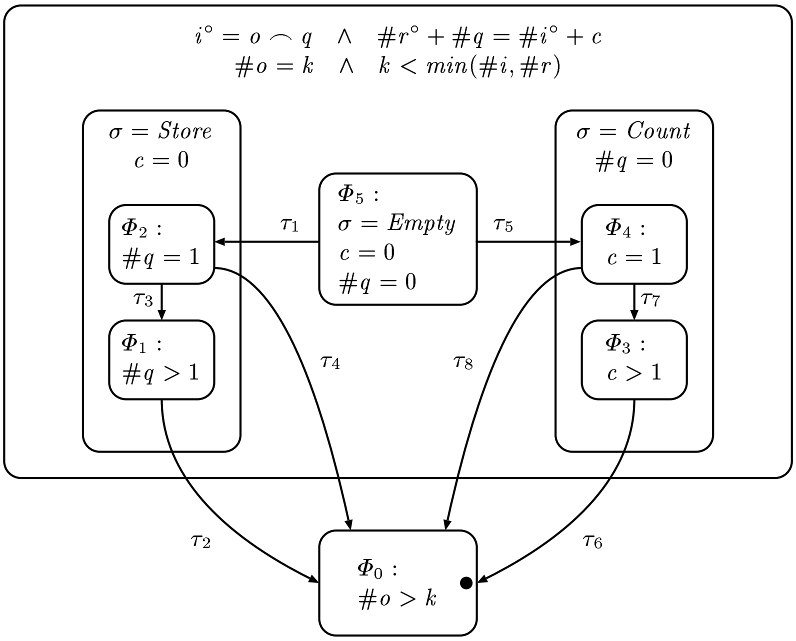

Fig. 5. Response Diagram for the Buffer

the enabledness of the transitions depend on the left hand side of the property, which implies that there is input waiting on i or r.

There are 50 proof obligations associated with our example response diagram. Similar to invariance diagrams, we need to show that the diagram is correct for all 5 nodes and all 9 transitions. Again, we do not represent loops or disabled transitions. Additionally, we have to show that at least one of the outgoing transitions of each node is enabled, leading to additional 5 proof obligations in this case. However, of the 50 obligations only 20 are non-trivial.

Using the proof rules of Section 3.2 we can derive black box properties from our results. From the invariant we can deduce properties of the buffer's histories. Note that since $i^\circ \sqsubseteq i$ and $r^\circ \sqsubseteq r$ and due to the admissability of the invariant we get

$$[\![Buffer]\!] \Rightarrow \big(o \sqsubseteq i \wedge \#o \leq \#r\big)$$

From the response property and with ℓ defined as $min(\#i, \#r)$ we can further derive

$$[\![Buffer]\!] \Rightarrow \#o \geq min(\#i, \#r)$$

This implies the expected property of the buffer:

$$[\![Buffer]\!] \Rightarrow o \sqsubseteq i \wedge \#o = min(\#i, \#r)$$

4.2 Tool Support

While the verification conditions associated with a verification diagram are simple, the size of property proofs is still quite formidable: A verification diagram with n nodes for a system with m transitions requires the verification of about $n \times m$ verification conditions. Obviously, without tool support property verification is not feasible. We formalized our approach in Isabelle/HOL as an extension of Shankar's PVS formalization of state machines [18] to handle liveness properties and asynchronous communication.

Given an Isabelle formalization of a state transition system, verification conditions are proven as theorems. Verification tactics assemble sets of verification conditions according to the structure of invariance and response diagrams to temporal logic formulas. The derivation of the history properties from the temporal logic properties is not handled within our formalization: This would require the use of the much more complicated logic of computable functions [16].

The Isabelle formalization is documented in [5]. The theory files and proof scripts can be accessed electronically [2]. Introductory texts to Isabelle are also available electronically [11].

5 Examples

This section gives a short overview of two examples for our approach. The first example is a model of the NetBill protocol [9, 19] for low-cost transactions of electronic goods in the internet. The main point of this example is to demonstrate that for each of the three system views of Section 2 there are correctness properties that are best expressed within this view. The formalization and verification of this example can be found in [4].

The second example is a simple communication system originally proposed by the DFKI [10]; it demonstrates the use of black box views for compositional verification of dataflow properties. It is treated in more detail, including a verification using Isabelle/HOL in [1, 5]. The Isabelle proof scripts are available online [2].

5.1 NetBill Protocol

Figure 6 shows the architecture of a NetBill system. It consists of an arbitrary number of customers, an arbitrary number of merchants and the centralized bank server. Transactions occur between a customer process, a merchant process, and a centralized bank server. All money-related activities occur at the bank server.

Figure 7 show a sample transaction of the NetBill protocol. The customer process c receives an order for electronic goods at a merchant m from the environment. It generates a unique transaction number which is used to identify the transaction in the subsequent message exchanges, and forwards the order to the merchant process. The merchant returns an invoice, which consists of a price statement and the encrypted goods. The customer process then issues a cheque

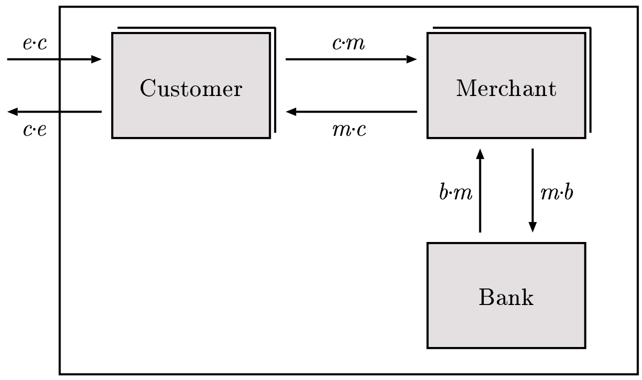

Fig. 6. NetBill architecture

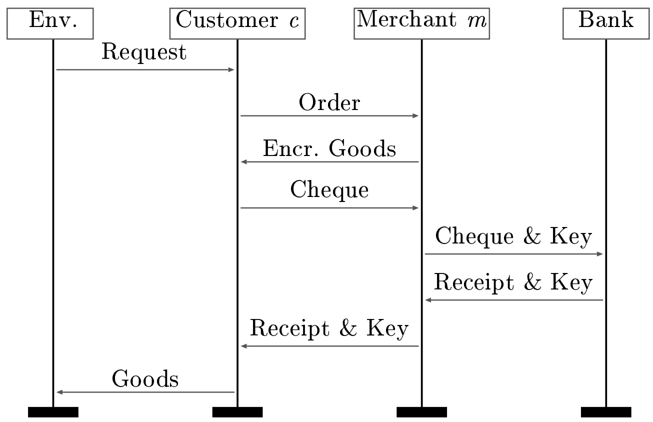

Fig. 7. NetBill Transaction

to the merchant, which states that it is willing to pay the price for the goods. This cheque is digitally clearsigned: Every participant in the protocol can read it, but it is impossible for anyone to change the information in it. This cheque, together with the key for decrypting the goods, is forwarded to the bank. The bank returns a receipt and the key to the merchant, which forwards it to the customer. With this key, the customer process decrypts the goods received earlier and delivers them to the user.

Customer, merchant and bank behavior are specified by rather simple state machines (see [4] for details). With this formalization, the following properties could be verified:

– *Guaranteed Delivery:* All goods ordered by the customer are delivered. Conversely, goods are only delivered if they were ordered.

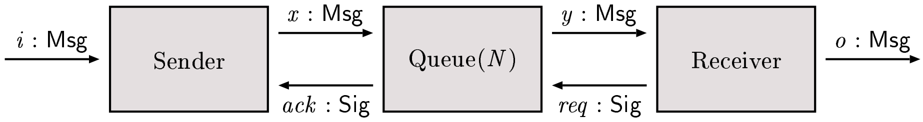

Fig. 8. Bounded Buffer

This property is expressed in the **black box view** of the system: It is a simple equality on subsequences of the messages transmitted over the system channels.

- *Guaranteed Payment:* For all goods ordered by a customer, the money amount corresponding to the good's price is subtracted from the customer account. This property is best expressed as a response property in the **execution view** of the system: Each state in an execution where a good is orderered is followed by a state where the price for the good is substracted from the customer account.
- *Money Atomicity:* The sum of the customer and merchant accounts remains invariant. Because of the centralized NetBill bank server, this property is quite obvious: It can be directly expressed as a predicate logic formula in the **state transition view** of the NetBill system.

5.2 Communication Queue

Figure 8 shows the system structure of a bounded transmission system with three components: a sender, a receiver, and a buffer with a capacity for $N \geq 2$ data messages. Operationally, the sender pushes data to the queue and waits for acknowledgments; the receiver pulls data from the queue by sending requests first. The queue itself stores up to N ($N \geq 2$) data messages.

The behavior of the three components is defined by simple state transition diagrams (omitted here, see [1]). With the proof techniques of Section 4, we can show that the components satisfy the following history relations:

$$\begin{array}{lll} [\![Sender]\!] & \Rightarrow & x \sqsubseteq i \\ & & \#x = min(\#i, 1 + \#ack) \\ [\![Queue(N)]\!] & \Rightarrow & y \sqsubseteq x \\ & & \#y \geq min(\#x, \#req) \\ & & \#ack = min(\#x, \#req + N - 1) \\ [\![Receiver]\!] & \Rightarrow & o \sqsubseteq y \\ & & \#o \geq \#y \\ & & \#req = 1 + \#y \end{array}$$

By black box composition, the history relation of the complete system is just the conjunction of the three component history relations above. From this conjunction is it immediate that the output is a prefix of the input: $o \sqsubseteq y \sqsubseteq x \sqsubseteq i$. By some case analysis on the communication history length inequalities it can also be shown that the length of the output equals the length of the input. Together, this implies

$$o = i$$

for all input streams i: As expected, the system implements the identity relation.

The same result could have been obtained by first composing the three component state machines, and then deriving $o \sqsubseteq i$ and $\#o \geq \#i$; the number of verification conditions for the invariance and response properties would have been much higher, however. For the composition of dataflow properties, history relations seem to be the more adequate abstraction level.

6 Conclusion

Both state based and history based views are useful in system design. Simple Hoare-like verification conditions, temporal logic formulas and history relations in the style of Focus [8] allow the formulation of properties at different abstraction levels to discuss the effects of single transitions, the structure of the components' reachable state space and the communication behavior within the system and to the environment. The main advantage of our approach is that it is straightforward to derive history based properties from simple temporal logic invariants and response properties.

Thus, the main effort remains in temporal logic proofs of state machine properties. Indeed, our Isabelle formalization tackles only this aspect; the step to black box properties and further reasoning about these black properties must be done by hand. While for the buffer example we were quite satisfied with the level of automatization that Isabelle provides, some first experiment with the Netbill protocol showed that here much additional work would be needed simply to reason about the data states of the components. It remains to be seen whether proof assistants that are more specialized for software and systems verification could offer more help in this respect.

Currently our work is based on a simple and standard state machine model and linear temporal logic to reason about the state based view. This choice is quite arbitrary: TLA or I/O automata together with the Temporal Logic of Steps [15] could also be used. Especially IOAs mights be a good choice to combine state based and history based system views for protocols.

Our specification and proof techniques are so far only suited for time independent systems with buffered directed communication. In principle, communication based on RPCs or transactions can be encoded in our model, but then the resulting proof obligations would be too detailed and removed from the original abstraction level. Instead, the general idea of streams as communication histories should be extended to more complex communication patterns. The extension of history-based specifications to time-dependent systems raises some interesting questions [6]. A straightforward solution might be to explicitly include "time ticks" in the message streams.

Acknowledgment. We thank the anonymous referees for their detailed comments.

References

1. M. Breitling and J. Philipps. Black Box Views of State Machines. Technical Report TUM-I9916, Institut für Informatik, TU München, 1999.
2. M. Breitling and J. Philipps. State machine theories and proof scripts for Isabelle/HOL. http://www.in.tum.de/~philipps/BBV, 2000.
3. M. Breitling and J. Philipps. Step by step to histories. In *AMAST 2000, LNCS 1816*, 2000.
4. M. Breitling and J. Philipps. Transitions into black box views - the NetBill protocol revisited. Technical Report TUM-I0013, Institut für Informatik, TU München, 2000.
5. M. Breitling and J. Philipps. Verification diagrams for dataflow properties. Technical Report TUM-I0005, Institut für Informatik, TU München, 2000.
6. M. Broy. Functional specification of time sensitive communicating systems. In *Models, Formalism, Correctness, LNCS 430*, pages 153–179, 1990.
7. M. Broy. From states to histories. In *Marktoberdorf Summer School on Engineering Theories of Software Construction*. Springer, NATO ASI Series F, 2000.
8. M. Broy and K. Stølen. *Specification and Development of Interactive Systems - FOCUS on Streams, Interfaces and Refinement*. Springer, 2001.
9. B. Cox, J. D. Tygar, and M. Sirbu. Netbill security and transaction protocol. In *Proc. of the First USENIX Workshop in Electronic Commerce*, pages 77–88, 1995.
10. D. Hutter, H. Mantel, G. Rock, and W. Stephan. Nebenläufige verteilte Systeme: Ein Producer/Consumer Beispiel. Internal Manuscript, DFKI, 1998.
11. Isabelle home page. http://isabelle.in.tum.de.
12. Z. Manna and A. Pnueli. *The Temporal Logic of Reactive and Concurrent Systems — Specification*. Springer, 1991.
13. Z. Manna and A. Pnueli. Models for reactivity. *Acta Informatica*, 30:609–678, 1993.
14. Z. Manna and A. Pnueli. Temporal verification diagrams. In *International Symposium on Theoretical Aspects of Computer Software, LNCS 789*, pages 726–765, 1994.
15. O. Müller. *A Verification Environment for I/O Automata Based on Formalized Meta-Theory*. PhD thesis, Technische Universität München, 1998.
16. L. C. Paulson. *Logic and Computation*. Cambridge University Press, 1987.
17. L. C. Paulson. *Isabelle: A Generic Theorem Prover, LNCS 828*. Springer, 1994.
18. N. Shankar. A lazy approach to compositional verification. Technical Report CSL-93-08, Computer Science Laboratory, SRI, 1993.
19. J. D. Tygar and M. Sirbu. Netbill: An internet commerce system optimized for network delivered services. *IEEE Personal Communications*, 2(4):34–39, 1995.