TECHNISCHE
UNIVERSITÄT
MÜNCHEN

INSTITUT FÜR INFORMATIK

**Sonderforschungsbereich 342:
Methoden und Werkzeuge für die Nutzung
paralleler Rechnerarchitekturen**

Black Box Views of State Machines

Max Breitling and Jan Philipps

**TUM-I9916
SFB-Bericht Nr. 342/07/99 A
Oktober 99**

TUM–INFO–10-I9916-100/1.–FI

Anforderungen an: Prof. Dr. A. Bode
Sprecher SFB 342
Institut für Informatik
Technische Universität München
D-80290 München, Germany

Druck: Fakultät für Informatik der
Technischen Universität München

Black Box Views of State Machines *

Max Breitling Jan Philipps

Institut für Informatik
Technische Universität München
D-80290 München

{breitlin|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, embedded systems and communication protocols. The techniques to reason about the dataflow within a system, however, are less well developed.

This report adapts a UNITY-like formalism for specification and verification to systems of asynchronously communicating components. The components themselves are specified as state machines. The resulting proof techniques allows abstract and compositional reasoning about dataflow properties of systems.

*This work was supported by the Sonderforschungsbereich 342 "Werkzeuge und Methoden für die Nutzung paralleler Rechnerarchitekturen" and by a travel grant of the Bayerische Forschungsstiftung.

Contents

1 Introduction

To allow precise reasoning about a hard- or software system, a mathematical foundation for both systems and properties is a prerequisite. For some classes of systems temporal logics have been used successfully to formalize and to reason about their properties. Prominent examples are circuit design and embedded systems software, where the clocked or cyclic operation model leads to a straightforward notion of a system state. The distinction between "allowed" and "forbidden" system states leads to natural invariance properties and proof techniques. Moreover, both hardware circuits and embedded software have essentially a finite state space, and exhaustive verification techniques, such as model checking, have been used with some success.

Temporal logic and model checking are less successful, however, when the dataflow between loosely coupled components that communicate asynchronously via communication channels is examined. Note that it is not simply a question of guessing an upper bound of the channel buffer size. Sometimes a system has a —for all purposes— unbounded buffer size. When examining an email-based groupware system, what would be the size of the internet between participating parties; and what would be the upper bound of the length of the list of unread mails?

For such systems, the state-based glass box view of a component is less useful than the black box view of its input and output. Black box properties of dataflow components and systems can be concisely formulated as relations over the communication history of components [12, 3, 13]; such properties are inherently modular and allow easy reasoning about the global system behavior given the component properties.

But also for data flow components a state-based glass box view can be helpful. State machines lead to natural proofs of safety properties by induction; they provide an operational intuition that can aid in finding ranking functions for some classes of liveness proofs; and finally, state machines are good design documents for a component's implementation.

In this report we show —based on the ideas of Broy's verification of the Alternating Bit Protocol [6]— how abstract specifications of the black box view of a system or system component can be combined with state machine-based descriptions of the system operations. Thus we combine techniques for easy verification of dataflow properties with descriptions that lead to efficient implementations of a system.

The property specification and verification technique is adapted from UNITY [27, 26]. The UNITY axioms for the safety and progress operators are proven to be correct in our mathematical model. Moreover, our framework allows compositional reasoning: Properties of single components can be used to deduce properties of the whole system.

This report is structured as follows: In Section 2 we introduce the mathematical basis for the black box view of components and systems; Section 3 introduces the abstract syntax and the semantics of state machines and their graphical representation, state transition diagrams. Safety and progress properties together with their proof techniques are described in Sections 4 and 5. The black box and state machine specifications are related in Section 6. In particular, it is shown how state machine properties can be used to derive properties of the state machine's black box view. Section 7 contains an extended example. The conclusion (Section 8) summarizes the results and contains an outlook on future work.

2 Black Box Specifications

A dataflow system is a network of components. Each component has input and output ports. The ports are connected by directed channels. The black box view regards only the communication between components and abstracts from the internal workings of the components.

FOCUS offers a mathematical basis for the black box view of dataflow systems. A detailed introduction into FOCUS can be found in [12, 3]. This section contains only a short overview over the concepts used in the rest of the report. The communication history of channels is represented by message sequences called *streams* (Section 2.1), components are modeled as relations between communication histories (Section 2.2), and systems are modeled as the composition of components (Section 2.3); component composition results again in a component.

2.1 Streams

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

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, $\#s = \infty$.
- $s \frown t$ denotes the concatenation of s and t. If s is infinite, $s \frown t = s$.
- $s \sqsubseteq t$ holds if s is a prefix of t:

  $$\exists\, u \in \mathsf{Msg}^\omega \bullet s \frown u = t$$

- s^j denotes the result of concatenating j copies of s; similarly, s^∞ results in the concatenation of infinitely many copies of s.
- $s.j$ is the j-th element of s, if $1 \leq j \leq \#s$, and is undefined otherwise.
- $s{\downarrow}j$ is the prefix of s with length j, if $0 \leq j \leq \#s$, and is undefined otherwise.
- $\mathsf{ft}.s$ denotes the first element of a stream, i.e. $\mathsf{ft}.s = s.1$, if $s \neq \langle\,\rangle$.
- $\mathsf{rt}.s$ is the stream s without the first element: $s = \langle \mathsf{ft}.s\rangle \frown \mathsf{rt}.s$ for all $s \neq \langle\,\rangle$.

The set of streams Msg^ω with the prefix order $\sqsubseteq$ forms a CPO with least element $\langle\,\rangle$. A *chain* is a set $\{\ s_i \ \mid\ i \in \mathbb{N}\ \}$ of streams, where for each i: $s_i \sqsubseteq s_{i+1}$. Since the set of streams is a CPO, each such chain has a unique least upper bound s which is denoted by

$$\bigsqcup\{\ s_i \ \mid\ i \in \mathbb{N}\}$$

The operators defined above as well as the notion of chains and least upper bounds can be extended pointwise to tuples of streams.

A function out of $\mathsf{Msg}_1^\omega \to \mathsf{Msg}_2^\omega$ is called a *continuous* function [28], iff

$$\bigsqcup\{\ f(s_i)\ \mid\ i \in \mathbb{N}\} = f\big(\bigsqcup\{\ s_i \ \mid\ i \in \mathbb{N}\}\big)$$

Continuous functions are also monotonic:

$$x \sqsubseteq y \Rightarrow f(x) \sqsubseteq f(y)$$

An example of a continuous function is the filter function Ⓢ; MⓈs is the substream of s that contains only messages also contained in the set M.

The filter function has the following properties:

$$\begin{array}{rll}
 & M\circledS\langle\,\rangle & = \langle\,\rangle \\
m \in M \Rightarrow & M\circledS(s \frown \langle m\rangle) & = (M\circledS s) \frown \langle m\rangle \\
m \notin M \Rightarrow & M\circledS(s \frown \langle m\rangle) & = (M\circledS s)
\end{array}$$

2.2 Components

Dataflow components are modeled as relations over communication histories. The relations are expressed using formulas in predicate logic where the formula's free variables range over streams. They represent the communication history over the component's input and output ports.

The black box behavior of a dataflow component U is specified by giving a set of input channel identifiers I_U, a set of output channel identifiers O_U (where $I_U \cap O_U = \varnothing$) and a predicate (for simplicity also denoted by U) with free variables from I_U and O_U. Each channel identifier has an assigned type that describes the set of messages allowed on that channel. We do not treat the typing of the identifiers formally in this paper.

Figure 1 shows a graphical representation of a component with two input channels i_1 and i_2 of type M_1 and M_2, where $M_1 \cap M_2 = \varnothing$, and a single output channel o of type $M = M_1 \cup M_2$. The intended black box behavior of this component is to merge the messages on the two input channels: all inputs received on the input channels are forwarded to the output channel o.

This behavior is specified with the sets for the input and output identifiers

$$I_{Merge} \stackrel{\mathsf{df}}{=} \{i_1, i_2\}, \quad O_{Merge} \stackrel{\mathsf{df}}{=} \{o\}$$

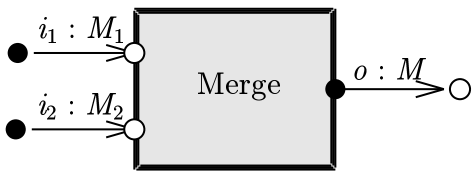

Figure 1: Component *Merge*

and the predicate

$$Merge \overset{\text{df}}{\Leftrightarrow} M_1 \circledS o = i_1 \ \wedge \ M_2 \circledS o = i_2$$

stating that the messages sent on o of type M_j are exactly the messages, in the same order, as received on channel i_j (for $j \in \{1, 2\}$).

Component behavior can be specified in the following more readable style:

Merge
in $i_1 : M_1, i_2 : M_2$
out $o : M$

$M_1 \circledS o = i_1$
$M_2 \circledS o = i_2$

Not all specifications in Focus are sensible: It is easy to specify *inconsistent* components by predicates that restrict the possible input histories of a component. A component is *realizable*, if it is possible to achieve its behavior step by step in a way that is causally correct. This means in particular that it is *monotonic*: It cannot take back messages that were sent earlier.

A detailed discussion of these requirements and their formalizations can be found in [15]. The specifications in this report are all consistent and realizable.

2.3 Black Box Composition

The black box view of a system can be derived from the black box views of the system's components by *composition*. Components may share input channels, but each output channel must be controlled by a single component. This is captured below in the definition of compatibility.

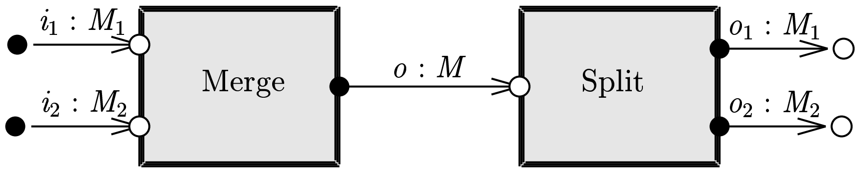

Figure 2: System *Multiplex*

Compatibility. Two components U and V are *compatible* if they do not share output channels:

$$O_U \cap O_V = \varnothing$$

Composition. Compatible components can be composed. The result of the composition $U \otimes V$ is again a component specification. Channels with identical names are connected, the output of the composition is the union of the two component's output channels as output, and the input of the composition consists of those input channels that remain unconnected.

$$I_{U \otimes V} \stackrel{\mathsf{df}}{=} (I_U \cup I_V) \setminus (O_U \cup O_V), \quad O_{U \otimes V} \stackrel{\mathsf{df}}{=} O_U \cup O_V$$

The system behavior is the conjunction of the component behavior predicates:

$$U \otimes V \stackrel{\mathsf{df}}{\Leftrightarrow} U \wedge V$$

Note that we decided to keep connected channels visible, so that no information is lost that could be useful in formal proofs. Although not all internals are hidden in our approach we still call it a black box view since the behavior is only described in an abstract way through a relation of streams, as opposed to state machines that use buffers, states and transitions as shown in the next section.

Figure 2 shows a system that models a multiplexing data transfer channel. In addition to the merge component, it consists of a split component, specified as:

Split

in $o : M$

out $o_1 : M_1, o_2 : M_2$

$o_1 = M_1 Ⓢ o$

$o_2 = M_2 Ⓢ o$

The composition of the two black box specifications is shown below:

Multiplex

in $i_1 : M_1, i_2 : M_2$
out $o : M, o_1 : M_1, o_2 : M_2$

$M_1 \circledS o = i_1$
$M_2 \circledS o = i_2$
$o_1 = M_1 \circledS o$
$o_2 = M_2 \circledS o$

2.4 Prefix Properties and Length Properties

A black box specification can be seen as a description of the input and output *properties* of a component. In practice, properties of dataflow systems are often expressed as a conjunction of equations

$$f(o) = F(i_1, \ldots i_n)$$

where $o \in O$ and F is a function that describes the output on o given input histories $i_1, \ldots i_n \in I$. Both f and F are assumed to be continuous. In many cases, f will just be the identity function.

Such equations can be split into a *prefix property*

$$f(o) \sqsubseteq F(i_1, \ldots i_n)$$

and a *length property*

$$\#f(o) \geq \#F(i_1, \ldots i_n)$$

For example, the specification of *Merge* can equivalently be formulated as follows:

Merge

in $i_1 : M_1, i_2 : M_2$
out $o : M$

$M_1 \circledS o \sqsubseteq i_1$
$M_2 \circledS o \sqsubseteq i_2$

$\#M_1 \circledS o \geq \#i_1$
$\#M_2 \circledS o \geq \#i_2$

Prefix properties are *safety* properties: Their violation can be detected as soon as an illegal output is produced. Length properties are *liveness* properties: Their violation cannot be detected by an observer, since it is always possible that the output is produced some time in the future. Liveness properties put demands on complete executions of a component, while safety properties restrict partial executions.

3 State Machines

In the previous section black box specifications of components and systems are introduced. With these more abstract specifications only the relation between complete input and output message streams is considered, but nothing is said about *how* the behavior of a component is achieved. In contrast, state machines describe a behavior in a stepwise manner.

In this section we show how dataflow components can be specified by state machines. We use the term state machine both for their abstract syntax (state transition systems, Section 3.2) and for their concrete graphical representation (state transition diagrams, Section 3.3). Section 3.4 defines the semantics of state machines, Section 3.5 their composition.

First we give a formal definition of variable valuations for an assertion. Variable valuations allow us to talk about the validity of assertions in the different states of a state machine execution.

3.1 Variable Valuations

Defining Var as the universe of all (unprimed) variables, we define a valuation α as a function that assigns to each variable in Var a value from the variable's type. By $\mathsf{free}(\Phi)$ we denote the set of free variables in a logical formula Φ. If an assertion Φ evaluates to true when each variable $v \in \mathsf{free}(\Phi)$ is replaced by $\alpha(v)$, we write

$$\alpha \models \Phi$$

Variable names can be *primed*: For example, v' is a new variable name that results from putting a prime behind v. We extend priming to sets

$$V' \stackrel{\mathsf{df}}{=} \{\ v' \ \mid\ v \in V\ \}$$

and to valuations: Given a valuation α of variables in Var, α' is a valuation of variables in V' with

$$\alpha'(v') = \alpha(v) \quad \text{for all variables } v \in \mathsf{Var}$$

Priming can also be extended to predicates, functions and other expressions: If Ψ is an assertion with $\mathsf{free}(\Psi) \subseteq V$, then Ψ' is the assertion that results from priming all free variables. Thus, $\mathsf{free}(\Psi') = (\mathsf{free}(\Psi))'$. Similarly, any expression expr' just denotes the expression expr with all variables primed.

Note that an unprimed valuation α assigns values to all *unprimed* variables, while a primed valuation β' only assigns values to all *primed* variables. If an assertion Φ contains both primed and unprimed variables, we need two valuations to determine its truth. If Φ evaluates to true, we write

$$\alpha, \beta' \models \Phi$$

Two valuations can *coincide* on a subset V of Var, defined as

$$\alpha \stackrel{V}{=} \beta \quad \stackrel{\mathsf{df}}{\Leftrightarrow} \quad \forall\, v \in V \bullet \alpha(v) = \beta(v)$$

3.2 State Transition Systems

A state transition system (STS) $\mathcal{S}$ is a tuple

$$(I, O, A, \mathcal{I}, \mathcal{T})$$

with the following components:

- I, O: Sets of input and output channel variables with $I \cap O = \varnothing$. Each variable in I and O ranges over finite streams. These variables hold the communication history from the environment to the component and from the component to its environment, respectively.

- A: A set of variables containing local state attributes of the STS (variables that hold the control state of the machine or some additional data) together with the set $\{\ i^\circ \ \mid\ i \in I\}$. We assume again $A \cap I = \varnothing = A \cap O$. The variables i° also range over finite streams; they stand for that part of the input i that has already been processed by the state machine. The part of i that has not been processed yet is denoted by i^+, and uniquely defined via

  $$i = i^\circ \frown i^+$$

  Since our state machines ensure that $i^\circ \sqsubseteq i$, i^+ is well-defined. Intuitively, i^+ is the implicit communication buffer of the asynchronous communication.

  We introduce $V \stackrel{\mathsf{df}}{=} I \cup O \cup A$ as the set containing all variables of an STS. Sometimes we refer to the variables in I as *external variables*, while we call the variables in $O \cup A$ *controlled variables*.

- $\mathcal{I}$: An assertion over the variables V which characterizes the set of initial states. We require that $\mathcal{I}$ is satisfiable.This means that there is a valuation α that satisfies the initial assertion and this validity does not depend on $i \in I$ nor is it restricting the values $i \in I$, i.e. the valuations for $i \in I$ can be exchanged arbitrarily:

  $$\exists\, \alpha \bullet \alpha \models \mathcal{I} \quad \wedge \quad \left(\forall\, \beta \bullet \beta \stackrel{O\cup A}{=} \alpha \quad \Rightarrow \quad \beta \models \mathcal{I}\right)$$

  Moreover, $\mathcal{I}$ asserts that initially no input has been processed:

  $$\mathcal{I} \Rightarrow i^\circ = \langle\,\rangle \quad \text{for all } i \in I$$

- $\mathcal{T}$: A finite set of transitions. Each transition $\tau \in \mathcal{T}$ is an assertion over the variables $V \cup V'$. The unprimed variables stand for variable valuations in the current state, the primed ones for valuations in the successor state.

  Each transition $\tau \in \mathcal{T}$ must fulfill the following requirements for all $i \in I, o \in O$. It may not take back messages it already sent, it may not undo the receipt of a message, it can only read what was sent to the component and the environment is not allowed to take back input:

  $$\tau \Rightarrow o \sqsubseteq o' \quad \wedge \quad i^\circ \sqsubseteq i^{\circ\prime} \quad \wedge \quad i^{\circ\prime} \sqsubseteq i \quad \wedge \quad i \sqsubseteq i'$$

  Transitions can only very weakly restrict the changes of the external variables I, since nothing should be assumed about the environment. The only constraint of the environment is that it may only *extend* the streams that are associated (by the valuations) with the variables in I, but in an arbitrary way. This can be formalized as follows: If a transition τ leads from state α to state β (valuations can be interpreted as states), this transition can contain some specific changes of the input variables. It must be shown that the same transition is also valid with an arbitrary extension of the input variables. This means that for all Val, β, γ:

  $$\alpha, \beta' \models \tau \quad \wedge \quad \beta \overset{O \cup A}{=} \gamma \quad \wedge \quad \forall\, i \in I \bullet \alpha(i) \sqsubseteq \gamma(i) \quad \Rightarrow \quad \alpha, \gamma' \models \tau$$

  In addition to the transitions in $\mathcal{T}$ there is always an implicit environment transition τ^ϵ which abstracts the possible behavior of the environment of $\mathcal{S}$. It is defined as follows:

  $$\tau^\epsilon \overset{\text{df}}{\Leftrightarrow} \bigwedge_{v \in O \cup A} v = v' \wedge \bigwedge_{i \in I} i \sqsubseteq i'$$

  The environment transition leaves all controlled variables unchanged, while the input variables may be extended. The environment transition, too, obeys the restrictions posed on the transitions in $\mathcal{T}$. The fairness of transitions is reflected in the definition of executions in Section 3.4.

Enabledness In Section 3.4, states of an STS are formalized as valuations for the variables V. Given a valuation α, we say that a transition τ is *enabled* in α, iff there is a valuation β for V such that

$$\alpha, \beta' \models \tau$$

We write

$$\alpha \models \mathsf{En}(\tau)$$

for the assertion that τ is enabled in α. Note that the environment transition τ^ϵ is enabled in every state (with $\beta = \alpha$).

To use the predicate in arbitrary (but of course well-formed) formulas, it can also be defined syntactically as follows:

$$\mathsf{En}(\tau) \stackrel{\mathsf{df}}{=} \exists\, v' \in \mathsf{Var}' \bullet \tau$$

If a transition is *not* enabled, we denote this as $\alpha \models \neg\, \mathsf{En}(\tau)$.

3.3 State Transition Diagrams

Typically, an STS is not specified by defining formally all elements of the quintuple, but by a *state transition diagram* (STD). We use a subset of the STD syntax from the AutoFocus CASE tool [14, 21]. The channel identifiers in I and O are not directly specified in an STD; they are taken from system structure diagrams, which describe component interfaces as well as component interconnection.

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 this vertex is marked 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:

$$\{\mathit{Precondition}\}\ \mathit{Inputs} \rhd \mathit{Outputs}\ \{\mathit{Postcondition}\}$$

Inputs and *Outputs* stand for lists of expressions of the form

$$i?x \quad \text{and} \quad o!\mathit{exp} \quad (i \in I, o \in O)$$

, respectively, where x is 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 equivalent to true, 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 is 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.

Example As an example, the merge component from the previous section (Figure 1) could be specified by the STD in Figure 3. The corresponding STS can be derived in a schematic way:

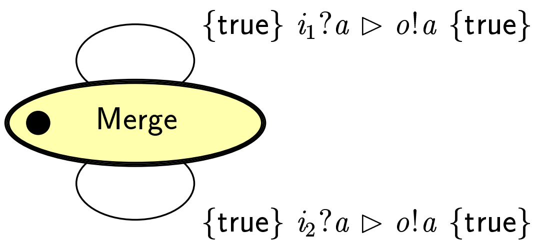

Figure 3: Merge STD

$$
\begin{aligned}
I &\stackrel{\text{df}}{=} \{i_1, i_2\}\\
O &\stackrel{\text{df}}{=} \{o\}\\
A &\stackrel{\text{df}}{=} \{\sigma, i_1^\circ, i_2^\circ\}\\
\mathcal{I} &\stackrel{\text{df}}{=} \sigma = \mathsf{Merge} \wedge i_1^\circ = \langle\,\rangle \wedge i_2^\circ = \langle\,\rangle \wedge o = \langle\,\rangle\\
\mathcal{T} &\stackrel{\text{df}}{=} \{\ \tau_1, \tau_2, \tau^\epsilon\ \}
\end{aligned}
$$

The variable σ holds the current control state of the STD. It is not really necessary here (since there is only one state) but included for illustration. There are no other internal variables. Note that $\mathcal{I}$ assures that initially no input is read and also it does not restrict the input variables, and therefore fulfills the necessary requirements. The transition τ_1 is defined by

$$
\begin{aligned}
\tau_1 \stackrel{\text{df}}{\Leftrightarrow}\ & \sigma = \mathsf{Merge} \wedge \sigma' = \mathsf{Merge} \wedge\\
& \exists\, a \in M_1 :\\
& \quad \mathsf{true} \wedge\\
& \quad \mathsf{ft}.i_1^+ = a \wedge i_1^{\circ\prime} = i_1^\circ \frown \langle a\rangle \wedge o' = o \frown \langle a\rangle \wedge\\
& \quad \mathsf{true} \wedge\\
& \quad i_2^{\circ\prime} = i_2^\circ \wedge\\
& \quad i_1 \sqsubseteq i_1' \wedge i_2 \sqsubseteq i_2'
\end{aligned}
$$

which states the following: The source and target state are both Merge, the empty precondition is trivially true, there is some message a that is available on channel i_1 and not yet read, which is then input and also sent on channel o. The postcondition is empty and therefore also trivially true. All variables not mentioned in the transition stay unchanged. In this case this results in a constant i_2°, meaning that we don't read on channel i_2. Finally, the environment can append arbitrary input to the input channels.

Note that the variable a in the transition label in Figure 3 does not appear as a variable in V. It is a transition-local variable.

Transition τ_2 is defined similarly by (now omitting "true" in the conjunction)

$$
\begin{aligned}
\tau_2 \overset{\mathrm{df}}{\Leftrightarrow}\ & \sigma = \mathsf{Merge} \wedge \sigma' = \mathsf{Merge} \wedge \\
& \exists\, a \in M_2 : \\
& \quad \mathsf{ft}.i_2^{+} = a \wedge i_2^{\circ\prime} = i_2^{\circ} \frown \langle a \rangle \wedge o' = o \frown \langle a \rangle \wedge \\
& \quad i_1^{\circ\prime} = i_1^{\circ} \wedge \\
& \quad i_1 \sqsubseteq i_1' \wedge i_2 \sqsubseteq i_2'
\end{aligned}
$$

The idle transition τ^{ϵ} allows the environment to sent messages to *Merge*, but keeps all variables unchanged that are under control of *Merge*.

$$
\begin{aligned}
\tau^{\epsilon} \overset{\mathrm{df}}{\Leftrightarrow}\ & \sigma' = \sigma \wedge \\
& i_1^{\circ\prime} = i_1^{\circ} \wedge i_2^{\prime\circ} = i_2^{\circ} \wedge o' = o \\
& i_1 \sqsubseteq i_1' \wedge i_2 \sqsubseteq i_2'
\end{aligned}
$$

In addition to the control state, components can have data state attributes. State attributes can be checked by the preconditions, and modified by the actions of a transition label, specified in the postcondition. The declaration of data state variables with their type and initialization can be supplied in an attached box in an STD, as shown in Figure 10.

A more detailed description of STDs, including extensions of STDs that allow hierarchical descriptions, can be found in [21, 11].

3.4 Executions

An *execution* of an STS $\mathcal{S}$ is an infinite stream ξ of valuations of the variables V that satisfies the following requirements:

- The first valuation in ξ satisfies the initialization assertion:

$$\xi.1 \models \mathcal{I}$$

- Each two subsequent valuations $\xi.k, \xi.(k+1)$ in ξ are related either by a transition in $\mathcal{T}$ or by the environment transition τ^{ϵ}:

$$\xi.k, \xi'.(k+1) \models \tau^{\epsilon} \vee \bigvee_{\tau \in \mathcal{T}} \tau$$

- Each transition $\tau \in \mathcal{T}$ of the STS is taken infinitely often in an execution, unless it is disabled infinitely often:

$$\big(\forall\, k \bullet \exists\, l \geq k \bullet \xi.l \models \neg\, \mathsf{En}(\tau)\big) \vee \big(\forall\, k \bullet \exists\, l \geq k \bullet \xi.l, \xi'.(l+1) \models \tau\big)$$

The set of executions of an STS $\mathcal{S}$ is denoted by $\langle\!\langle \mathcal{S} \rangle\!\rangle$; it is defined by

$$
\begin{aligned}
\langle\!\langle \mathcal{S} \rangle\!\rangle \stackrel{\text{df}}{=} \{\ \xi \ \mid\ & \xi.1 \models \mathcal{I} \wedge \\
& \forall\, k \in \mathbb{N} : \xi.k, \xi'.(k+1) \models \tau^{\epsilon} \vee \bigvee_{\tau \in \mathcal{T}} \tau \wedge \\
& \bigwedge_{\tau \in \mathcal{T}} \Big(\big(\forall\, k \bullet \exists\, l \geq k \bullet \xi.l \models \neg\, \mathsf{En}(\tau) \big) \vee \\
& \qquad \big(\forall\, k \bullet \exists\, l \geq k \bullet \xi.l, \xi'.(l+1) \models \tau \big) \Big)\ \}
\end{aligned}
$$

By induction it is easy to show that for each state $\xi.k$ in an execution

$$\xi.k \models i^{\circ} \sqsubseteq i$$

holds. Moreover, changes in the valuations of input variables I, output variables O and the processed input variables $\{ i^{\circ} \mid i \in I \}$ in subsequent states are restricted to the prefix order $\sqsubseteq$.

3.5 State Machine Composition

State machines can be composed if they are compatible. Similar to the compatibility of black box specifications, two state machines $\mathcal{S}_1 = (I_1, O_1, A_1, \mathcal{I}_1, \mathcal{T}_1)$ and $\mathcal{S}_2 = (I_2, O_2, A_2, \mathcal{I}_2, \mathcal{T}_2)$ are compatible if their controlled variables are disjoint and if there is no conflict concerning internal variables, i.e. no machine may access the internal variables of the other machine:

$$
\begin{aligned}
&(O_1 \cup A_1) \cap (O_2 \cup A_2) = \varnothing \wedge \\
&A_1 \cap I_2 = \varnothing \wedge A_2 \cap I_1 = \varnothing
\end{aligned}
$$

Thus, the two components may only share variables which are input variables of at least one of the two components. In Figure 4 the separation of the variables in different disjoint sets is visualized for a composition of two state machines. All messages on channels i with $i \in I_1 \cap I_2$ can be read by both machines independently. In order to ensure that the variables from A_1 and A_2 are disjoint in this case, the variables i° have to be renamed to i_1° or i_2° throughout the variable sets, transitions and initialization predicates of $\mathcal{S}_1$ and $\mathcal{S}_2$, respectively.

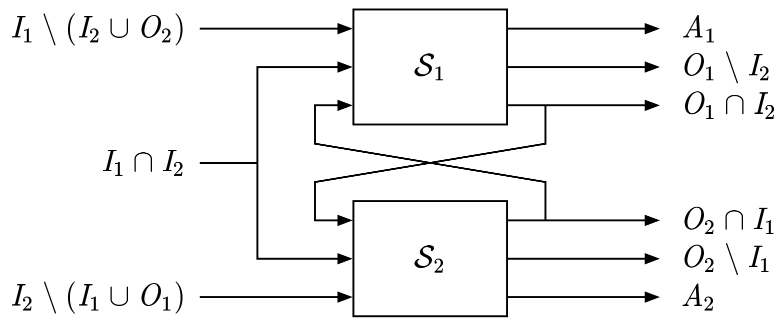

Figure 4: Composition of State Machines

The composition $\mathcal{S} = \mathcal{S}_1 \| \mathcal{S}_2$ is defined as the STS with the following components:

$$
\begin{aligned}
I &\stackrel{\text{df}}{=} (I_1 \cup I_2) \setminus (O_1 \cup O_2) \\
O &\stackrel{\text{df}}{=} O_1 \cup O_2 \\
A &\stackrel{\text{df}}{=} A_1 \cup A_2 \\
\mathcal{I} &\stackrel{\text{df}}{\Leftrightarrow} \mathcal{I}_1 \wedge \mathcal{I}_2 \\
\mathcal{T} &\stackrel{\text{df}}{=} \{ \tau_1 \wedge \tau_2^\epsilon \mid \tau_1 \in \mathcal{T}_1 \} \cup \{ \tau_1^\epsilon \wedge \tau_2 \mid \tau_2 \in \mathcal{T}_2 \} \\
\tau^\epsilon &\stackrel{\text{df}}{\Leftrightarrow} \bigwedge_{v \in O \cup A} v = v' \wedge \bigwedge_{i \in I} i \sqsubseteq i'
\end{aligned}
$$

From the definition above, it is easy to see that composition is associative:

$$\mathcal{S}_1 \| (\mathcal{S}_2 \| \mathcal{S}_3) = (\mathcal{S}_1 \| \mathcal{S}_2) \| \mathcal{S}_3$$

The resulting STS satisfies the requirements of Section 3.2:

Proof:

- The variable sets of $\mathcal{S}$ satisfy the disjointness and inclusion requirements posed on I, O and A.
- $\mathcal{I}$ fulfills the requirements

  $$\exists \alpha \bullet \alpha \models \mathcal{I} \quad \wedge \quad \left(\forall \beta \bullet \beta \stackrel{O \cup A}{=} \alpha \quad \Rightarrow \quad \beta \models \mathcal{I} \right)$$

  and

  $$\mathcal{I} \Rightarrow i^\circ = \langle \rangle \quad \text{for all } i \in I$$

  what is proved here:

  - $\mathcal{I}$ is indeed satisfiable for all input valuations of I: With $\mathcal{S}_1$ and $\mathcal{S}_2$ as proper STS, we know that there exist α_1 and α_2 with

    $$
    \begin{aligned}
    &\alpha_1 \models \mathcal{I}_1 \quad \wedge \quad \forall \beta \bullet \beta \stackrel{O_1 \cup A_1}{=} \alpha_1 \quad \Rightarrow \quad \beta \models \mathcal{I}_1 \\
    &\alpha_2 \models \mathcal{I}_2 \quad \wedge \quad \forall \beta \bullet \beta \stackrel{O_2 \cup A_2}{=} \alpha_2 \quad \Rightarrow \quad \beta \models \mathcal{I}_2
    \end{aligned}
    $$

and we have to show

$$\exists\, \alpha \bullet \alpha \models \mathcal{I} \quad \wedge \quad \forall\, \beta \bullet \beta \overset{O \cup A}{=} \alpha \quad \Rightarrow \quad \beta \models \mathcal{I}$$

We now define the α we are looking for on the controlled variables of $\mathcal{S}$ by

$$\alpha(v) = \begin{cases} \alpha_1(v) & \text{if} \quad v \in O_1 \cup A_1 \\ \alpha_2(v) & \text{if} \quad v \in O_2 \cup A_2 \end{cases}$$

For $v \in I$ we allow any valuation. Note that this definition is conflict-free since $(O_1 \cup A_1) \cap (O_2 \cup A_2) = \varnothing$ and $I \cap (A_1 \cup A_2) = \varnothing$. Hence

$$\alpha \overset{O_1 \cup A_1}{=} \alpha_1 \wedge \alpha \overset{O_2 \cup A_2}{=} \alpha_2$$

and therefore (instantiating β with α)

$$\alpha \models \mathcal{I}_1 \wedge \alpha \models \mathcal{I}_2 \quad \Leftrightarrow \quad \alpha \models \mathcal{I}_1 \wedge \mathcal{I}_2 \quad \Leftrightarrow \quad \alpha \models \mathcal{I}$$

For some β with $\beta \overset{O \cup A}{=} \alpha$ we have $\beta \overset{O_1 \cup A_1}{=} \alpha$ and $\beta \overset{O_2 \cup A_2}{=} \alpha$ due to the subset relations, so we conclude

$$\beta \models \mathcal{I}_1 \wedge \mathcal{I}_2$$

– The second requirement for $\mathcal{I}$ is also fulfilled since all input valuations are initially empty:

$$\mathcal{I} \Rightarrow \mathcal{I}_1 \wedge \mathcal{I}_2 \Rightarrow (\forall\, i_1 \in I_1 \bullet i_1^\circ = \langle\,\rangle) \wedge (\forall\, i_2 \in I_2 \bullet i_2^\circ = \langle\,\rangle) \Rightarrow \forall\, i \in I \bullet i^\circ = \langle\,\rangle$$

- The set $\mathcal{T}$ of the composed system has all properties it should have:
  – All transitions τ allow only restricted changes of the channel valuations:

$$\tau \Rightarrow o \sqsubseteq o' \wedge i^\circ \sqsubseteq i^{\circ\prime} \wedge i^{\circ\prime} \sqsubseteq i' \wedge i \sqsubseteq i'$$

The empty transition τ^ϵ trivially fulfills this requirement. All other transitions τ consist of $\tau_1 \wedge \tau_2^\epsilon$ or $\tau_2 \wedge \tau_1^\epsilon$. Because of this symmetry, we show the proof only for the first case.

$$\begin{aligned} \tau \quad &\Leftrightarrow \quad \tau_1 \wedge \tau_2^\epsilon \\ &\Rightarrow \quad \forall\, o_1 \in O_1, i_1 \in I_1 \bullet o_1 \sqsubseteq o_1' \wedge i_1^\circ \sqsubseteq i_1^{\circ\prime} \wedge i_1^{\circ\prime} \sqsubseteq i_1' \wedge i_1 \sqsubseteq i_1' \wedge \\ &\qquad \bigwedge_{o_2 \in O_2} o_2 = o_2' \wedge \bigwedge_{i_2 \in I_2} i_2^\circ = i_2^{\circ\prime} \wedge \bigwedge_{i_2 \in I_2} i_2 \sqsubseteq i_2' \end{aligned}$$

Note that quantifying over $i \in I$ and over $i^\circ \in A$ ranges over the same variables in any occurrence of i°, according to the definition of A. By rearranging the terms we reach

$$\begin{aligned} &\forall\, o_1 \in O_1 \bullet o_1 \sqsubseteq o_1' \wedge \forall\, o_2 \in O_2 \bullet o_2 = o_2' \\ &\forall\, i_1^\circ \in A_1 \bullet i_1^\circ \sqsubseteq i_1^{\circ\prime} \wedge \forall\, i_2^\circ \in A_2 \bullet i_2^\circ = i_2^{\circ\prime} \wedge \\ &\forall\, i_1 \in I_1 \bullet i_1 \sqsubseteq i_1' \wedge \forall\, i_2 \in I_2 \bullet i_2 \sqsubseteq i_2' \end{aligned}$$

and therefore

$$\forall\, o \in O \bullet o \sqsubseteq o' \;\wedge\; \forall\, i^\circ \in A \bullet i^\circ \sqsubseteq i^{\circ\prime} \;\wedge\; \forall\, i \in I \bullet i \sqsubseteq i'$$

Finally, with $i_2^{\circ\prime} = i_2^\circ \overset{(ax)}{\sqsubseteq} i_2 \sqsubseteq i_2'$ for all $i_2 \in I_2$ and $\forall\, i_1 \in I_1 \bullet i_1^{\circ\prime} \sqsubseteq i_1'$ we get

$$\forall\, i \in I \bullet i^{\circ\prime} \sqsubseteq i'$$

- Additionally, we show (again for only one of the two cases) that τ is restricting the variables in I in the proper way, i.e. we assume

$$\alpha, \beta' \models \tau_1 \wedge \tau_2^\epsilon \tag{1}$$
$$\beta \overset{O\cup A}{=} \gamma \tag{2}$$
$$\forall\, i \in I_1 \cup I_2 \bullet \alpha(i) \sqsubseteq \gamma(i) \tag{3}$$

  and show

$$\alpha, \gamma' \models \tau_1 \wedge \tau_2^\epsilon$$

  $\alpha, \gamma' \models \tau_1$ follows directly from the properties of τ_1 with the subset relations $O_1 \cup A_1 \subseteq O \cup A$ and $I_1 \subseteq I$. It remains to show

$$\alpha, \gamma' \models \bigwedge_{v \in O_2 \cup A_2} v = v' \wedge \bigwedge_{i_2 \in I_2} i_2 \sqsubseteq i_2'$$

  Assuming $v \in O_2 \cup A_2$, we prove the first half by

$$\alpha(v) \overset{(1)}{=} \beta'(v') \overset{(Def.)}{=} \beta(v) \overset{(2)}{=} \gamma(v) \overset{(Def.)}{=} \gamma'(v')$$

  Finally, with

$$i_1 \in I_1 \Rightarrow \alpha(i_1) \overset{(3)}{\sqsubseteq} \gamma(i_1) \quad \Rightarrow \quad \alpha, \gamma' \models i_1 \sqsubseteq i_1'$$

  we conclude the proof.

- There is always an environment transition τ^ϵ that is defined exactly as in Section 3.2.

• We also prove here that composition maintains the enabledness of transitions, i.e. it holds

$$\alpha \models \mathsf{En}(\tau_1) \quad \Leftrightarrow \quad \alpha \models \mathsf{En}(\tau_1 \wedge \tau_2^\epsilon)$$
$$\alpha \models \mathsf{En}(\tau_2) \quad \Leftrightarrow \quad \alpha \models \mathsf{En}(\tau_2 \wedge \tau_1^\epsilon)$$

We only show the first property, and prove for an arbitrary α only that

$$\exists\, \beta \bullet \;\alpha, \beta' \models \tau_1 \quad \Rightarrow \quad \exists\, \gamma \bullet \;\alpha, \gamma' \models \tau_1 \wedge \tau_2^\epsilon$$

The opposite direction is obvious. For some β we define γ by

$$\gamma(v) = \begin{cases} \alpha(v) & \text{if} \quad v \in (O_2 \cap I_1) \cup (A_2 \cup (O_2 \setminus I_1)) \cup ((I_1 \cup I_2) \setminus (O_1 \cup O_2)) \\ \beta(v) & \text{if} \quad v \in (A_1 \cup (O_1 \setminus I_2)) \cup (O_1 \cap I_2) \end{cases}$$

With this definition we have

$$\gamma \overset{A_1 \cup O_1}{=} \beta \quad \text{and} \quad \forall\, v \in I_1 \bullet \alpha(v) \sqsubseteq \gamma(v)$$

and therefore (due to the properties of τ_1)

$$\alpha, \gamma' \models \tau_1$$

For $v \in O_2 \cup A_2$ we have $\gamma(v) = \alpha(v)$, so

$$\alpha, \gamma' \models \bigwedge_{v \in O_2 \cup A_2} v = v'$$

is valid. For $v \in I_2 \cap O_1$ we know $\beta(v) = \gamma(v)$, and from the assumption $\alpha, \beta' \models \tau_1$ that $\alpha(v) \sqsubseteq \beta'(v')$, since $\tau_1 \Rightarrow v \sqsubseteq v'$ ($v \in O_1$!), and this leads to $\alpha(v) \sqsubseteq \gamma(v)$. For $v \in I_2 \setminus O_1$ we defined $\gamma(v) = \alpha(v)$, so that $\alpha, \gamma' \models v = v'$. Together, we have

$$\alpha, \gamma' \models \bigwedge_{i_2 \in I_2} i_2 \sqsubseteq i_2'$$

and therefore

$$\alpha, \gamma' \models \tau_2$$

which finishes the proof.

□

The main property of the composed system is that the runs of $\mathcal{S}$ are subsets of the runs of $\mathcal{S}_1$ and $\mathcal{S}_2$:

$$\langle\!\langle \mathcal{S}_1 \| \mathcal{S}_2 \rangle\!\rangle \subseteq \langle\!\langle \mathcal{S}_1 \rangle\!\rangle \quad \text{and} \quad \langle\!\langle \mathcal{S}_1 \| \mathcal{S}_2 \rangle\!\rangle \subseteq \langle\!\langle \mathcal{S}_2 \rangle\!\rangle$$

The converse does not hold: Since each component may restrict the input to the other component, some executions of the individual components may not be possible after composition.

Proof: We only need to show that each $\xi \in \langle\!\langle \mathcal{S}_1 \| \mathcal{S}_2 \rangle\!\rangle$ is also in $\langle\!\langle \mathcal{S}_1 \rangle\!\rangle$; the proof for $\mathcal{S}_2$ is symmetrical. Expanding the definition of $\langle\!\langle . \rangle\!\rangle$, we have to show:

1. If $\xi.1 \models \mathcal{I}$, then also $\xi.1 \models \mathcal{I}_1$.
2. For all $k \in \mathbb{N}$

$$\xi.k, \xi'.(k+1) \models \tau^\epsilon \vee \bigvee_{\tau \in \mathcal{T}} \tau \quad \Rightarrow \quad \xi.k, \xi'.(k+1) \models \tau_1^\epsilon \vee \bigvee_{\tau_1 \in \mathcal{T}_1} \tau_1$$

3. For all $\tau \in \mathcal{T}$, if

$$\big(\forall\, k \bullet \exists\, l \geq k \bullet \xi.l \models \neg\, \mathsf{En}(\tau)\big) \vee \big(\forall\, k \bullet \exists\, l \geq k \bullet \xi.l, \xi.(l+1) \models \tau\big)$$

then for all $\tau_1 \in \mathcal{T}_1$

$$\big(\forall\, k \bullet \exists\, l \geq k \bullet \xi.l \models \neg\, \mathsf{En}(\tau_1)\big) \vee \big(\forall\, k \bullet \exists\, l \geq k \bullet \xi.l, \xi.(l+1) \models \tau_1\big)$$

The proof for the initialization assertion is immediate, since $\mathcal{I} \Rightarrow \mathcal{I}_1$.

The proof of the consecution assertion distinguishes the kind of transition that $\mathcal{S}$ makes:

- If $\mathcal{S}$ makes the environment transition τ^ϵ, $\mathcal{S}_1$ also makes its environment transition τ_1^ϵ.

  This is valid since $\tau^\epsilon \Rightarrow \tau_1^\epsilon$: All inputs of $\mathcal{S}_1$ are either also inputs of $\mathcal{S}$ (and thus can only be extended), or outputs of $\mathcal{S}_2$ (and thus left unchanged). The other variables of $\mathcal{S}_1$ are left unchanged.

- If $\mathcal{S}$ makes a transition that consists of the environment transition τ_1^ϵ of $\mathcal{S}_1$ and a proper transition $\tau_2 \in \mathcal{T}_2$, $\mathcal{S}_1$ also makes an environment transition.

  This is valid since the controlled variables of $\mathcal{S}_1$ remain unchanged, the environment inputs can only be extended, and the inputs connected to outputs of $\mathcal{S}_2$ also can only be extended by τ_2.

- If $\mathcal{S}$ makes a transition that consists of a proper transition $\tau_1 \in \mathcal{T}$ and the environment transition τ_2^ϵ of $\mathcal{S}_2$, $\mathcal{S}_1$ makes the transition τ_1.

For the fairness assumption, it is sufficient to show the following two properties for each $\tau_1 \in \mathcal{T}_1$:

$$\big(\forall\, k \bullet \exists\, l \bullet \xi.l \models \neg\, \mathsf{En}(\tau_1 \wedge \tau_2^\epsilon)\big) \Rightarrow \big(\forall\, k \bullet \exists\, l \bullet \xi.l \models \neg\, \mathsf{En}(\tau_1)\big)$$

and

$$\big(\forall\, k \bullet \exists\, l \geq k \bullet \xi.l, \xi.(l+1) \models (\tau_1 \wedge \tau_2^\epsilon)\big) \Rightarrow \big(\forall\, k \bullet \exists\, l \geq k \bullet \xi.l, \xi.(l+1) \models \tau_1\big)$$

If we identify k and l in the consequences of the implications with the k and l on the left-hand-sides, respectively, the proofs are immediate. $\square$

4 Safety Properties

This section introduces proof principles for safety properties of a state machine. A typical safety property is invariance, which means that an assertion over the variables of the state machine holds in every state of every execution.

The proof principles for safety properties are inspired by the UNITY theory [27]. In Section 4.1 the *constrains* operator **co** of UNITY is adapted to our state machine framework; Section 4.2 introduces some verification rules. In particular, the axioms of UNITY's **co** are shown to be valid in our framework.

4.1 Predicates and Properties

State machine properties are expressed using assertions that relate communication histories and the values of the attribute variables.

A *state predicate* of a state machine $\mathcal{S} = (I, O, A, \mathcal{I}, \mathcal{T})$ is an assertion Φ where the free variables range over the variables in $V = I \cup O \cup A$; a *history predicate* is a formula where the free variables range only over the input and output variables $I \cup O$.

An example for a state predicate is the initialization assertion $\mathcal{I}$ of the state machine. Below is the initialization assertion of the state machine *Merge* (Figure 3):

$$\sigma = \mathsf{Merge} \wedge i_1^+ = i_1 \wedge i_1^\circ = \langle\,\rangle \wedge i_2^+ = i_2 \wedge i_2^\circ = \langle\,\rangle \wedge o = \langle\,\rangle$$

An example for a history predicate is the black box specification of *Merge* (Section 2.2):

$$M_1 \circledS o = i_1 \wedge M_2 \circledS o = i_2$$

State predicates relate the communication histories and state variables at a given point in a system execution. To express properties about the complete execution, predicates are lifted to executions by one of the following two operators:

- **initially** Φ holds for a state machine $\mathcal{S}$ and a state predicate Φ, iff Φ is true under the variable valuation of the first time point of each system run:

  $$\forall\, \xi \in \langle\!\langle \mathcal{S} \rangle\!\rangle \bullet \xi.1 \models \Phi$$

  This is denoted by $\mathcal{S} \models$ **initially** Φ. It holds if the characterization of the initial states imply Φ, i.e. if $\mathcal{I} \Rightarrow \Phi$ is valid.

- Φ **co** Ψ holds for a state machine $\mathcal{S}$ and state predicates Φ and Ψ (Φ *constrains* Ψ), iff whenever Φ evaluates to true at a point in a system execution, then so does Ψ at the subsequent point:

  $$\forall\, \xi \in \langle\!\langle \mathcal{S} \rangle\!\rangle \bullet \forall\, k \bullet \big(\xi.k \models \Phi \Rightarrow \xi.(k+1) \models \Psi\big)$$

  This is denoted by $\mathcal{S} \models \Phi$ **co** Ψ. The operator **co** is defined to have a weaker binding than all other logical operators.

We also use the following abbreviations:

$$\mathcal{S} \models \textbf{stable}\ \ \Phi \stackrel{\text{df}}{\Leftrightarrow} \mathcal{S} \models \Phi\ \textbf{co}\ \Phi$$

$$\mathcal{S} \models \textbf{inv}\ \Phi \qquad \stackrel{\text{df}}{\Leftrightarrow} \mathcal{S} \models \textbf{stable}\ \ \Phi \text{ and } \mathcal{S} \models \textbf{initially}\ \Phi$$

Informally, a predicate is stable if its validity is preserved by all transitions of a system, and we call it an invariant, if it holds in all reachable states.

The abbreviations can also be expressed at the level of executions: By induction on k it can be shown that

$$\mathcal{S} \models \textbf{inv}\ \Phi \quad \Leftrightarrow \quad \forall\, \xi \in \langle\!\langle \mathcal{S} \rangle\!\rangle \bullet \forall\, k \bullet \xi.k \models \Phi$$

This characterization can be used to lift tautologies and general results from the state machine level to the property level. If we have

$$\mathcal{S} \models \textbf{inv}\ \Phi$$

and know

$$\forall\, v \in \mathsf{free}(\Phi) \cup \mathsf{free}(\Psi) \bullet \quad \Phi \Rightarrow \Psi$$

we can also deduce

$$\mathcal{S} \models \textbf{inv}\ \Psi$$

As another example, we know that in every execution $\xi \in \langle\!\langle \mathcal{S} \rangle\!\rangle$ and for every input $i \in I$ we have $\forall\, k \bullet \xi.k \models i^\circ \sqsubseteq i$ We can apply this knowledge in property proofs since it can be lifted to $\mathcal{S} \models \textbf{inv}\ i^\circ \sqsubseteq i$.

Free variables can be introduced to relate the left and the right side of a **co** property. This technique can be used to lift the fact that outputs are only extended within an execution, either expressed directly with the output variable, or with the output variable's length:

$$\mathcal{S} \models x = o\ \textbf{co}\ x \sqsubseteq o$$

$$\mathcal{S} \models \#o = k\ \textbf{co}\ \#o \geq k$$

Free variables as x and k are universally qualified (comparable to so-called *rigid* variables). In the semantics, the first of the two properties above is denoted by

$$\forall\, \xi \in \langle\!\langle \mathcal{S} \rangle\!\rangle, k \in \mathbb{N}, x \in X \bullet \big(\xi.k \models x = o \Rightarrow \xi'.(k+1) \models x \sqsubseteq o'\big)$$

where X is the type of the variable x (which should be the same as the type of the state machine output variable o).

Note that $\Phi \Rightarrow \Psi$ cannot be concluded from $\Phi\ \textbf{co}\ \Psi$, as the following counterexample shows: Assume a state machine $\mathcal{S}$ with a integer variable x, that is initialized with the value 0 and that can only be incremented. Obviously we have *true* **co** $x \geq 0$, while *true* $\Rightarrow x \geq 0$ is not valid. Properties using **co** have only to be valid for all reachable states, while an implication must be true for all valuations of the variables.

$$\begin{array}{|l}
\mathcal{I} \Rightarrow \Phi \\
\hline
\mathcal{S} \models \textbf{initially } \Phi
\end{array}$$

(a) Initiality

$$\begin{array}{|l}
\Phi \wedge \tau^{\epsilon} \Rightarrow \Psi' \\
\Phi \wedge \tau \Rightarrow \Psi' \quad \text{for all } \tau \in \mathcal{T} \\
\hline
\mathcal{S} \models \Phi \textbf{ co } \Psi
\end{array}$$

(b) Consecution

$$\begin{array}{|l}
\mathcal{S} \models \Phi_1 \textbf{ co } \Psi_1 \\
\mathcal{S} \models \Phi_2 \textbf{ co } \Psi_2 \\
\hline
\mathcal{S} \models \Phi_1 \wedge \Phi_2 \textbf{ co } \Psi_1 \wedge \Psi_2 \\
\mathcal{S} \models \Phi_1 \vee \Phi_2 \textbf{ co } \Psi_1 \vee \Psi_2
\end{array}$$

(c) Conjunction and Disjunction

$$\begin{array}{|l}
\mathcal{S} \models \Phi \textbf{ co } \Psi \\
\mathcal{S} \models \Psi \textbf{ co } \chi \\
\hline
\mathcal{S} \models \Phi \textbf{ co } \chi
\end{array}$$

(d) Transitivity

$$\begin{array}{|l}
\mathcal{S} \models \Phi \textbf{ co } \Psi \\
\hline
\mathcal{S} \models \Phi \wedge \chi \textbf{ co } \Psi
\end{array}$$

(e) LHS Strengthening

$$\begin{array}{|l}
\mathcal{S} \models \Phi \textbf{ co } \Psi \\
\hline
\mathcal{S} \models \Phi \textbf{ co } \Psi \vee \chi
\end{array}$$

(f) RHS Weakening

Figure 5: Basic rules for **co**

4.2 Verification Rules

With suitable verification rules it is possible to verify system properties from the abstract syntax of state machines, without having to expand the definitions of **initially** and **co** and to verify properties semantically.

In this section, we state and prove some verification rules.

4.2.1 Basic Rules

Figure 5 shows a number rules that are frequently used in verification:

The initialization and consecution rules (Figure 5(a), 5(b)) lift the semantic definitions of **initially** Φ and Φ **co** Ψ to the level of the abstract syntax of state machines. The conjunction and disjunction rule (Figure 5(c)) combine two **co**-properties into one. Note the similarity of this rule —as well as the strengthening and weakening rules in Figure 5(e), 5(f)— to the usual rules for logical implication. Indeed, as Misra remarks [27], the **co** operator is a kind of temporal implication. Finally, **co** is transitive (Figure 5(d)).

Proof:
Initialization. The validity of the initialization rule follows immediately from the definition of **initially** Φ.

Consecution. For the consecution rule, assume that $\xi \in \langle\!\langle \mathcal{S} \rangle\!\rangle$ and that $\xi.k \models \Phi$ for an arbitrary $k \in \mathbb{N}$. Then,

$$\xi.k, \xi'.(k+1) \models \Phi$$

since $\mathsf{free}(\Phi) \subseteq V$, i.e. Φ contains no primed variables. From the definition of executions, we know that

$$\xi.k, \xi'.(k+1) \models \tau^{\epsilon} \vee \bigvee_{\tau \in \mathcal{T}} \tau$$

Assume now that

$$\xi.k, \xi'.(k+1) \models \tau^{\epsilon}$$

and therefore

$$\xi.k, \xi'.(k+1) \models \Phi \wedge \tau^{\epsilon}$$

From the first premise we conclude

$$\xi.k, \xi'.(k+1) \models \Psi'$$

On the other hand, if for a $\tau \in \mathcal{T}$

$$\xi.k, \xi'.(k+1) \models \tau$$

we know

$$\xi.k, \xi'.(k+1) \models \Phi \wedge \tau$$

Because of the second premise also

$$\xi.k, \xi'.(k+1) \models \Psi'$$

Since $\mathsf{free}(\Psi') \subseteq V'$,

$$\xi'(k+1) \models \Psi'$$

therefore

$$\xi.(k+1) \models \Psi$$

which concludes the proof.

Conjunction and Disjunction. We just show the conjunction part of the rule. Assume that

$$\xi.k \models \Phi_1 \wedge \Phi_2$$

or, equivalently

$$\xi.k \models \Phi_1 \quad \text{and} \quad \xi.k \models \Phi_2$$

From the premises we conclude

$$\xi.(k+1) \models \Psi_1 \quad \text{and} \quad \xi.(k+1) \models \Psi_2$$

and thus

$$\xi.(k+1) \models \Psi_1 \wedge \Psi_2$$

The proof of the disjunction part is similar. The rules for the strengthening of the left hand side and weakening of the right hand side of **co** are corollaries of the conjunction and disjunction rule.

Transitivity rule. The validity of the rule is not obvious: The conclusion relates two consecutive states, as do the two premises. Intuitively, then, χ should hold not in the state following Φ, but in the one after that. The rule is proven by introducing a *stuttering step* via an additional environment transition into an execution.

Given arbitrary $\xi \in \langle\langle \mathcal{S} \rangle\rangle$ and $k \in \mathbb{N}$ such that $\xi.k \models \Phi$, we need to show that

$$\xi.(k+1) \models \chi$$

First, we construct a sequence $\widehat{\xi}$ from ξ by repeating the k-th state of ξ:

$$\widehat{\xi}.l = \begin{cases} \xi.l & \text{if} \quad l \leq k \\ \xi.k & \text{if} \quad l = k+1 \\ \xi.(l-1) & \text{if} \quad l > k+1 \end{cases}$$

The repetition of the state $\xi.k$ corresponds to an environment transition τ^ϵ which leaves the external variables unchanged; hence $\widehat{\xi}$, too, is an execution of $\mathcal{S}$.

Now,

$$\begin{array}{lll} \xi.k \models \Phi & \Rightarrow \; \widehat{\xi}.k \models \Phi, & \text{since } \widehat{\xi}.k = \xi.k \\ & \Rightarrow \; \widehat{\xi}.(k+1) \models \Psi, & \text{since } \mathcal{S} \models \Phi \;\mathbf{co}\; \Psi \\ & \Rightarrow \; \widehat{\xi}.(k+2) \models \chi, & \text{since } \mathcal{S} \models \Psi \;\mathbf{co}\; \chi \\ & \Rightarrow \; \xi.(k+1) \models \chi, & \text{since } \widehat{\xi}.(k+2) = \xi.(k+1) \end{array}$$

□

4.2.2 Invariant Substitution Rules

The two rules in Figure 6 are related to UNITY's substitution axiom; they are taken from Paulson's Isabelle formalization of UNITY [30].

$$\begin{array}{|l} \mathcal{S} \models \mathbf{inv}\,\chi \\ \mathcal{S} \models \Phi \wedge \chi \;\mathbf{co}\; \Psi \\ \hline \mathcal{S} \models \Phi \;\mathbf{co}\; \Psi \end{array}$$

(a) LHS Invariant Elimination

$$\begin{array}{|l} \mathcal{S} \models \mathbf{inv}\,\chi \\ \mathcal{S} \models \Phi \;\mathbf{co}\; \Psi \\ \hline \mathcal{S} \models \Phi \;\mathbf{co}\; \Psi \wedge \chi \end{array}$$

(b) RHS Invariant Introduction

Figure 6: Invariant substitution rules

The first rule allows us to remove invariants on the left hand side, while the second one allows us to to introduce invariants on the right hand side. The proofs of these rules are shown below. Invariant introduction on the left side and invariant elimination on the right side is also possible: This can be handled by the strengthening and weakening rules of Figure 5.

Proof:
Invariant elimination on the left side. We need to show that if

$$(1) \quad \forall\, \xi \in \langle\!\langle \mathcal{S} \rangle\!\rangle, k \bullet \xi.k \models \chi$$

and

$$(2) \quad \forall\, \xi \in \langle\!\langle \mathcal{S} \rangle\!\rangle, k \bullet \xi.k \models \Phi \wedge \chi \Rightarrow \xi.(k+1) \models \Psi$$

then for arbitrary $\xi \in \langle\!\langle \mathcal{S} \rangle\!\rangle$, k:

$$\xi.k \models \Phi \Rightarrow \xi.(k+1) \models \Psi$$

After instantiating the quantifiers in (2), we have

$$\xi.k \models \Phi \wedge \chi \Rightarrow \xi.(k+1) \models \Psi$$

hence

$$\xi.k \models \chi \wedge \xi.k \models \Phi \Rightarrow \xi.(k+1) \models \Psi$$

or, equivalently,

$$\xi.k \models \chi \Rightarrow \big(\xi.k \models \Phi \Rightarrow \xi.(k+1) \models \Psi\big)$$

Instantiating (1), we have

$$\xi.k \models \chi$$

and therefore

$$\xi.k \models \Phi \Rightarrow \xi.(k+1) \models \Psi$$

which concludes the proof.

Invariant introduction on the right side. We need to show that if

$$(1) \quad \forall\, \xi \in \langle\!\langle \mathcal{S} \rangle\!\rangle, k \bullet \xi.k \models \chi$$

and

$$(2) \quad \forall\, \xi \in \langle\!\langle \mathcal{S} \rangle\!\rangle, k \bullet \xi.k \models \Phi \Rightarrow \xi.(k+1) \models \Psi$$

then for arbitrary $\xi \in \langle\!\langle \mathcal{S} \rangle\!\rangle$, k:

$$\xi.k \models \Phi \Rightarrow \xi.(k+1) \models \Psi \wedge \chi$$

After instantiating the quantifiers in (2) and (1), we have

$$\xi.k \models \Phi \Rightarrow \xi.(k+1) \models \Psi \quad \text{and} \quad \xi.(k+1) \models \chi$$

Thus,

$$\xi.k \models \Phi \Rightarrow \big(\xi.(k+1) \models \Psi \wedge \xi.(k+1) \models \chi\big)$$

which concludes the proof. □

4.3 Example

As an example of how to use the verification rules in practice, we continue our example of Section 3.3 and show that the state machine of *Merge* (Figure 3) only produces output that has been received on its input channels before:

We need to show that

$$\mathit{Merge} \models \mathbf{inv}\ M_1 \circledS o \sqsubseteq i_1 \quad \text{and} \quad \mathit{Merge} \models \mathbf{inv}\ M_2 \circledS o \sqsubseteq i_2$$

Note that this is not identical to the prefix properties of the Merge component as formulated in Section 2.4; here we only show that the prefix properties hold in every state of an execution, not that it holds for the complete I/O behavior of an execution. In Section 6, we show how to relate complete I/O behavior with invariants.

The two properties are symmetrical; we just show the one for i_1. With $i_1 = i_1^\circ \frown i_1^+$ we know that

$$(M_1 Ⓢ o) = i_1^\circ \quad \Rightarrow \quad (M_1 Ⓢ o) \sqsubseteq i_1$$

so that according to the observation of Section 4.1 it suffices to show

$$\mathit{Merge} \models \mathbf{inv}\ M_1 Ⓢ o = i_1^\circ$$

According to the definition of **inv** and the verification rules for **initially** and **co** this expands to the following proof obligations:

$$\mathcal{I} \Rightarrow M_1 Ⓢ o = i_1^\circ \tag{1}$$
$$M_1 Ⓢ o = i_1^\circ \wedge \tau^\epsilon \Rightarrow M_1 Ⓢ o' = i_1^{\circ\prime} \tag{2}$$
$$M_1 Ⓢ o = i_1^\circ \wedge \tau_1 \Rightarrow M_1 Ⓢ o' = i_1^{\circ\prime} \tag{3}$$
$$M_1 Ⓢ o = i_1^\circ \wedge \tau_2 \Rightarrow M_1 Ⓢ o' = i_1^{\circ\prime} \tag{4}$$

Proof: Since $\mathcal{I}$ implies $o = i_1^\circ = \langle\,\rangle$, (1) is trivial. The idle transition τ^ϵ implies $i^{\circ\prime} = i^\circ$ and $o' = o$, so that (2) is also obvious.

Expanding τ_1 in (3) shows its validity:

$$\begin{aligned}
&M_1 Ⓢ o' = i_1^{\circ\prime} \Leftrightarrow \\
&M_1 Ⓢ (o \frown \langle a \rangle) = i_1^\circ \frown \langle a \rangle \overset{a \in M_1}{\Leftrightarrow} \\
&(M_1 Ⓢ o) \frown \langle a \rangle = i_1^\circ \frown \langle a \rangle \Leftrightarrow \\
&M_1 Ⓢ o = i_1^\circ
\end{aligned}$$

Using the definition of τ_2 in (4) completes the proof:

$$\begin{aligned}
&M_1 Ⓢ o' = i_1^{\circ\prime} \Leftrightarrow \\
&M_1 Ⓢ (o \frown \langle a \rangle) = i_1^\circ \overset{a \notin M_1}{\Leftrightarrow} \\
&M_1 Ⓢ o = i_1^\circ
\end{aligned}$$

$\square$

4.4 Compositionality

From the compositionality result of Section 3.5 we can derive the following rules:

$$\frac{\mathcal{S}_1 \models \mathbf{initially}\ \Phi}{\mathcal{S}_1 \| \mathcal{S}_2 \models \mathbf{initially}\ \Phi} \qquad \frac{\mathcal{S}_1 \models \Phi\ \mathbf{co}\ \Psi}{\mathcal{S}_1 \| \mathcal{S}_2 \models \Phi\ \mathbf{co}\ \Psi}$$

Proof: Let ξ be an arbitrary execution from $[\![\mathcal{S}_1 \| \mathcal{S}_2]\!]$. From Section 3.5 we know that then $\xi \in [\![\mathcal{S}_1]\!]$ also holds. The premises of the two rules imply

$$\xi.1 \models \Phi$$

and

$$\forall\, k \bullet \big(\xi.k \models \Phi \Rightarrow \xi.(k+1) \models \Psi\big)$$

Since ξ is also a run of $[\![\mathcal{S}_1 \| \mathcal{S}_2]\!]$, this also means

$$\mathcal{S}_1 \| \mathcal{S}_2 \models \textbf{initially}\ \Phi$$

and

$$\mathcal{S}_1 \| \mathcal{S}_2 \models \Phi\ \textbf{co}\ \Psi$$

$\square$

A corollary of the above rules is that every invariant of a system remains an invariant after composition:

$$\frac{\mathcal{S}_1 \models \textbf{inv}\ \Phi}{\mathcal{S}_1 \| \mathcal{S}_2 \models \textbf{inv}\ \Phi}$$

The compositionality of initiality, constraints and invariants is due to the dataflow structure of our systems: Components interact only by the transmission of messages, and since the arrival of new messages cannot disable component transitions, components cannot interfere in an unexpected way.

In UNITY, components can interfere; hence, compositionality rules like the ones above are not valid in general.

5 Progress Properties

Safety properties are useful to show that the system does not enter an illegal state or output illegal data. It is easy to build a system that fulfills safety properties: A system that simply does nothing fulfills any safety property.

To ensure that a system indeed processes its input and produces output, progress properties are used. As for safety properties, the progress properties are related to UNITY [26]. They are based on a "leadsto" operator $\mapsto$ (Section 5.1).

5.1 Leads-To Properties

Progress is expressed by the leadsto operator $\mapsto$. Intuitively, $\Phi \mapsto \Psi$ means that whenever in a state machine execution a state is reached where Φ holds, at the same or at a later point in the execution a state is reached where Ψ holds.

The semantic definition of $\mathcal{S} \models \Phi \mapsto \Psi$ is as follows. For all $\xi \in \langle\!\langle \mathcal{S} \rangle\!\rangle$,

$$\forall\, k \bullet (\xi.k \models \Phi) \Rightarrow (\exists\, l \geq k \bullet \xi.l \models \Psi)$$

From the semantic definition it follows immediately that $\mapsto$ is transitive, and that whenever $\Phi \Rightarrow \Psi$, then also $\Phi \mapsto \Psi$.

5.2 Verification Rules

For the leadsto operator there is also a set of verification rules so that properties can be shown at the level of state transition systems without reasoning about the system executions themselves.

5.2.1 Basic Rules

Figure 7 shows a number rules that are frequently used in verification. Essentially, leadsto properties are proved as follows. With the ensure rule (Figure 7(a)) leadsto properties that relate states that are separated by only a single proper transition are shown; this transition is called the *helpful* transition. From this basis, more elaborate properties are derived by the transitivity (Figure 7(b)) and disjunction (Figure 7(c)) rules.

Rule 7(f) (RHS weakening) is a special case of Rule 7(c) (disjunction). It would be sufficient for finite state systems; the disjunction rule is needed to show properties of infinite state systems (see [26] for a detailed explanation).

Proof:
Ensure. The proof is by contradiction. Assume that the premises of the rule in Fig-

$$
\begin{array}{|l}
\mathcal{S} \models \Phi \wedge \neg\, \Psi \ \mathbf{co}\ \Phi \vee \Psi \\
\text{For a transition } \tau \in \mathcal{T}\text{:} \\
\qquad \Phi \wedge \neg\, \Psi \Rightarrow \mathsf{En}(\tau) \\
\qquad\qquad \text{and} \\
\qquad \Phi \wedge \neg\, \Psi \wedge \tau \Rightarrow \Psi' \\
\hline
\mathcal{S} \models \Phi \mapsto \Psi
\end{array}
$$

(a) Ensure

$$
\begin{array}{|l}
\mathcal{S} \models \Phi \mapsto \Psi \\
\mathcal{S} \models \Psi \mapsto \chi \\
\hline
\mathcal{S} \models \Phi \mapsto \chi
\end{array}
$$

(b) Transitivity

$$
\begin{array}{|l}
\mathcal{S} \models \Phi(x) \mapsto \Psi \quad \text{for all } x \in X \\
\hline
\mathcal{S} \models \big(\exists\, x \in X \bullet \Phi(x)\big) \mapsto \Psi
\end{array}
$$

(c) Disjunction

$$
\begin{array}{|l}
\Phi \Rightarrow \Psi \\
\hline
\mathcal{S} \models \Phi \mapsto \Psi
\end{array}
$$

(d) Implication

$$
\begin{array}{|l}
\mathcal{S} \models \Phi \mapsto \Psi \\
\hline
\mathcal{S} \models \Phi \wedge \chi \mapsto \Psi
\end{array}
$$

(e) LHS Strengthening

$$
\begin{array}{|l}
\mathcal{S} \models \Phi \mapsto \Psi \\
\hline
\mathcal{S} \models \Phi \mapsto \Psi \vee \chi
\end{array}
$$

(f) RHS Weakening

Figure 7: Basic rules for $\mapsto$

ure 7(a) hold, but not its conclusion. Then there is an execution $\xi \in \langle\!\langle \mathcal{S} \rangle\!\rangle$ and a $k \in \mathbb{N}$ with

$$\xi.k \models \Phi$$

but for all $l \geq k$

$$\xi.l \models \neg\,\Psi$$

In particular, then

$$\xi.k \models \neg\,\Psi$$

and, by induction and the first premise, for all $n \geq k$:

$$\xi.n \models \Phi \wedge \neg\,\Psi$$

By the first part of the second premise, there is a $\tau \in \mathcal{T}$ such that for all $n \geq k$

$$\xi.n \models \mathsf{En}(\tau)$$

Because of the fairness assumption of state machine executions (Section 3.4), this means that there is an $m \geq k$ such that

$$\xi.m, \xi'.(m+1) \models \tau$$

and thus

$$\xi.m, \xi'.(m+1) \models \Phi \wedge \neg\,\Psi \wedge \tau$$

Because of the second part of the second premise, this implies

$$\xi.m, \xi'.(m+1) \models \Psi'$$

hence

$$\xi.(m+1) \models \Psi$$

which contradicts the assumption that there is no $l \geq k$ which validates Ψ.

Transitivity and implication. These rules are immediate consequences of the semantic definition of $\mapsto$ (Section 5.1).

Disjunction. Let X be an arbitrary set, $\Phi(x)$ a shear of state predicates parameterized by $x \in X$. Assume that $\xi \in \langle\!\langle \mathcal{S} \rangle\!\rangle$, and k and x such that

$$\xi.k \models \Phi(x)$$

From the premise we know that

$$\exists\, l \geq k \bullet \xi.l \models \Psi$$

Hence, the rule is valid.

Strengthening and weakening. The validity of these rules follows immediately from the semantic definition of $\mapsto$. □

5.2.2 Induction Rule

Non-trivial progress proofs often make use of some kind of a ranking function or measure, based on well-founded orders. This is formalized in the following rule. Let $(W, \prec)$ be a well-founded order, m a variable that ranges over W, and M a W-valued expression with free variables from V.

$$\frac{\mathcal{S} \models \big(p \wedge M = m\big) \mapsto \big(p \wedge M < m\big) \vee q \quad \text{for all } m \in W}{\mathcal{S} \models p \mapsto q}$$

The validity proof of this rule in analogous to the proof of this rule in [27].

5.2.3 Invariant Substitution Rules

The UNITY substitution axiom holds for leadsto properties as well; the four rules in Figure 8 correspond to the substitution rules for **co** in Section 4.2.

$$\frac{\begin{array}{l}\mathcal{S} \models \mathbf{inv}\,\chi \\ \mathcal{S} \models \Phi \wedge \chi \mapsto \Psi\end{array}}{\mathcal{S} \models \Phi \mapsto \Psi}$$

(a) LHS Invariant Elimination

$$\frac{\begin{array}{l}\mathcal{S} \models \mathbf{inv}\,\chi \\ \mathcal{S} \models \Phi \mapsto \Psi\end{array}}{\mathcal{S} \models \Phi \mapsto \Psi \wedge \chi}$$

(b) RHS Invariant Introduction

Figure 8: Invariant substitution rules

The proof of these rules is analogous to the proof of the corresponding rules in Section 4.2. Again, invariant introduction on the left side and invariant elimination on the right side follow immediately from the strengthening and weakening rules.

5.2.4 Output Extension Rule

The typical application for leadsto properties in dataflow systems is to show that a component produces output. Such properties can be formalized using the following property pattern:

$$\mathcal{S} \models \#o = k \wedge k < \ell \mapsto \#o > k$$

where $o \in O$ is an output variable of the component, and ℓ is a $\mathbb{N}$-valued expression with $\mathsf{free}(\ell) \subseteq I \cup O$ that is monotonic in the values of its free variables.

For output extension, the ensure rule can be further simplified. For the first premise of ensure, we need to show

$$\begin{aligned}\mathcal{S} \models \#o = k \wedge k < \ell \wedge \#o \leq k \\ \textbf{co } \big(\#o = k \wedge k < \ell\big) \vee \#o > k\end{aligned}$$

By predicate logic, this is equivalent to

$$\mathcal{S} \models \#o = k \wedge k < \ell \textbf{ co } \#o \geq k \wedge \big(k < \ell \vee \#o > k\big)$$

We now show that in dataflow systems this property always holds. For all variables $v \in I \cup O$,

$$(1) \quad \mathcal{S} \models \#v = k \textbf{ co } \#v \geq k$$

(see Section 4.1). This implies in particular that ℓ cannot become smaller because ℓ is monotone, i.e.

$$(2) \quad \mathcal{S} \models k < \ell \textbf{ co } k < \ell$$

By LHS strengthening (1) with $k < \ell$ we obtain

$$\mathcal{S} \models \#o = k \wedge k < \ell \textbf{ co } \#o \geq k$$

Similarly, we strengthen the LHS of (2) with $\#o = k$ and weaken its RHS with $\#o > k$:

$$\mathcal{S} \models \#o = k \wedge k < \ell \textbf{ co } k < \ell \vee \#o > k$$

These two properties can be combined with the conjunction rule of **co** to yield the first premise of the ensure rule.

Thus, for output extension, the following rule is already sufficient:

$$\begin{array}{|l}
\text{For a transition } \tau \in \mathcal{T}\text{:} \\
\qquad \#o = k \wedge k < \ell \Rightarrow \mathsf{En}(\tau) \\
\qquad\qquad \text{and} \\
\qquad \#o = k \wedge k < \ell \wedge \tau \Rightarrow \#o' > k \\
\hline
\mathcal{S} \models \#o = k \wedge k < \ell \mapsto \#o > k
\end{array}$$

Note that this is a quite substantial reduction in practice: It reduces the number of verification conditions from $n + 3$ to 2, where n is the number of transitions in $\mathcal{T}$.

The output extension rule is still valid, when o is replaced by $f(o)$, where f is a function that is monotonic according to the prefix order $\sqsubseteq$.

Another useful variation is the following rule, where the left hand sides of the $\mapsto$ operator are strengthened by Φ.

$$
\begin{array}{|l}
\text{For a transition } \tau \in \mathcal{T}\text{:} \\
\qquad \Phi \wedge \#o = k \wedge k < \ell \Rightarrow \mathsf{En}(\tau) \\
\qquad\qquad \text{and} \\
\qquad \Phi \wedge \#o = k \wedge k < \ell \wedge \tau \Rightarrow \#o' > k \\
\hline
\mathcal{S} \models \Phi \wedge \#o = k \wedge k < \ell \mapsto \#o > k
\end{array}
$$

5.3 Example

To demonstrate the verification rules for $\mapsto$, we continue the example of Section 3.3 and Section 4.3.

In Section 4.3, we learned that

$$Merge \models \mathbf{inv}\ M_1 \circledS o = i_1^\circ$$

and therefore

$$Merge \models \mathbf{inv}\ M_1 \circledS o \sqsubseteq i_1$$

This is a pure safety property: no M_1-output is emitted by the merge component, that has not been received on i_1 before; moreover, the order of the messages from M_1 on i_1 and o is identical.

This property also holds for a component that never reads from its input channels and never outputs anything on its output channel.

However, for the merge component we can show that

$$Merge \models \#M_1 \circledS o = k \wedge \#i_1 > k \mapsto \#M_1 \circledS o > k$$

Informally, this property means that whenever data is available on the input channel i_1, the component will at some time output further data of type M_1 on its output channel. Note that this does not mean that the output on o is indeed the same data that the component received from i_1; this has already been shown by the safety property of Section 4.3.

Proof: In this case, it is sufficient to just use the output extension rule. The rule has two premises, where we have to choose a transition $\tau \in \mathcal{T}$. The obvious choice is transition τ_1.

- For showing the first premise, we assume

$$\#M_1 \circledS o = k \wedge k < \#i_1$$

and have to show that τ_1 is enabled, i.e. we need values for the primed variables that evaluate τ_1 to true. From Section 4.3 we know $(M_1 Ⓢ o) = i_1^\circ$, so we have

$$\#i_1 = \#(i_1^\circ \frown i_1^+) = \#i_1^\circ + \#i_1^+ = \#(M_1 Ⓢ o) + \#i_1^+ = k + \#i_1^+$$

Therefore we have with $k < \#i_1$

$$\#i_1^+ > 0 \Rightarrow \exists\, a \in M_1 \bullet \mathsf{ft}.i_1^+ = a$$

The values for the remaining primed variables can be chosen according to τ_1.

- The second premise states

$$\#(M_1 Ⓢ o) = k \wedge k < \#i_1 \wedge \tau_1 \Rightarrow \#(M_1 Ⓢ o') > k$$

and is easy to show:

$$\#(M_1 Ⓢ o') = \#(M_1 Ⓢ o \frown \langle a \rangle) \overset{a \in M_1}{=} \#(M_1 Ⓢ o) + 1 = k + 1 > k$$

Note that we only showed that the output is eventually produced when input is available on i_1. This does not necessarily imply that indeed all input from i_1 appears on o; in Section 6 this gap is closed.

The proof that $M_2 Ⓢ o$ is extended when messages are available on channel i_2 is analogous. □

5.4 Compositionality

Leadsto properties are compositional. The validity of the following rule follows from the compositionality result from Section 3.5; the proof is similar to the compositionality result of Section 4.4.

$$\frac{\mathcal{S}_1 \models \Phi \mapsto \Psi}{\mathcal{S}_1 \| \mathcal{S}_2 \models \Phi \mapsto \Psi}$$

6 Black Box Views of State Machines

Both safety and liveness properties of state machines are based on state and history predicates. These predicates relate communication histories up to a time point and attribute values at this time point.

Typical dataflow properties cannot be expressed in this way. For example, the *Merge* component property that all input of channel i_1 is forwarded to the output is a property about the complete state machine execution, and not of the individual states in the execution.

This section closes the gap between state machines and black box views that describe the I/O behavior of a system for complete executions. In Section 6.1 the black box view of a state machine is defined; Sections 6.2 and 6.3 show how safety and liveness properties of a state machine can be used to deduce properties of its black box view.

6.1 Black Box Views

Within a state machine execution ξ, 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, 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.

The *black box view* of a state machine $\mathcal{S} = (I, O, A, \mathcal{I}, \mathcal{T})$ is 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:

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

Since both the proper transitions $\tau \in \mathcal{T}$ and the environment transition τ^ϵ of a state machine allow arbitrary extension of the input variable valuations, it is possible to successively approximate every possible input history. This means that the black box view $[\![\mathcal{S}]\!]$ is *complete* with respect to the input variables of $\mathcal{S}$: For an arbitrary input there is always some reaction of the system. Formally, this reads as: For each valuation α for the variables $I \cup O$ there exists a valuation β for $I \cup O$ such that

$$\alpha \stackrel{I}{=} \beta \quad \text{and} \quad \beta \in [\![\mathcal{S}]\!]$$

6.2 Safety Properties

In practice, it is difficult to directly use the black box semantics of a state machine defined in Section 6.1. Instead, we deduce properties about the black box view from properties of the state machine. Technically, a property of the black box view $[\![\mathcal{S}]\!]$ is a history predicate Φ (see Section 4.1) which is valid for each valuation in a system's black box view:

$$\forall\, \alpha \in [\![\mathcal{S}]\!] \bullet \alpha \models \Phi$$

We then write $[\![\mathcal{S}]\!] \Rightarrow \Phi$.

A useful class of history predicates is that of *admissible* predicates [28]. A history predicate Φ is admissible in a set of variable $W \subseteq \mathsf{free}(\Phi)$ if it holds for the limit of a chain of valuations for its variables, provided that it holds for each element of the chain. If predicate Φ is admissible in $\mathsf{free}(\Phi)$ it is simply called admissible. The free variables in a history predicate all range over the CPO of streams; the concepts of chain and limit are taken from Section 2.1).

If Φ is an admissible invariant history property of a state machine, it holds not only in every state of a system run, but also for the complete communication history:

$$\begin{array}{|l}
\mathsf{free}(\Phi) \subseteq I \cup O \\
\mathsf{adm}\ \Phi \\
\mathcal{S} \models \mathbf{inv}\, \Phi \\
\hline
[\![\mathcal{S}]\!] \Rightarrow \Phi
\end{array}$$

Proof: Expanding the definition of $\mathcal{S} \models \mathbf{inv}\, \Phi$, we have

$$\forall\, \xi \in \langle\!\langle \mathcal{S} \rangle\!\rangle, k \in \mathbb{N} \bullet \xi.k \models \Phi$$

In other words, Φ holds when its free variables v (where $v \in I \cup O$) are replaced by

$$\xi.k(v)$$

for each $k \in \mathbb{N}$. Since Φ is admissible, it also holds when its free variables are replaced by the least upper bounds

$$\xi.\infty(v)$$

This implies the conclusion of the rule. $\square$

It is in general not trivial to show the admissibility of a given property. However, Paulson gives in [28] some simple syntactical criteria for admissibility. For example, conjunctions and disjunctions of the following expressions over streams s, t, u are admissible in both s and t, but not in u:

$$\begin{array}{ll} s = t & \#s = \#t \\ s \sqsubseteq t & \#s \leq \#t \\ u \sqsubset s & \#u < \#s \end{array}$$

Here s, t, u need not be simple stream variables or constants; they can also be terms built from continuous functions (according to the prefix order $\sqsubseteq$), because admissibility is compositional through continuous functions.

Example

In Section 4.3, the following invariant property of the merge component has been derived:

$$\mathcal{S} \models M_1 \circledS o \sqsubseteq i_1 \wedge M_2 \circledS o \sqsubseteq i_2$$

Since this property is admissible —see above—, the following black box property of the component holds:

$$[\![Merge]\!] \Rightarrow M_1 \circledS o \sqsubseteq i_1 \wedge M_2 \circledS o \sqsubseteq i_2$$

6.3 Liveness Properties

In general, progress properties expressed with the leadsto operator $\mapsto$ cannot be lifted to complete executions. Still, from output extension properties (Section 5.2), liveness properties of a state machine's black box view can be derived.

Let ℓ be an $\mathbb{N}$-valued expression with $\mathsf{free}(\ell) \subseteq I$ that is monotonic in the values of its free variables, and $o \in O$.

$$\begin{array}{|l} \mathsf{free}(\ell) \subseteq I \cup O \\ \mathcal{S} \models \#o = \nu \wedge \nu < \ell \mapsto \#o > \nu \\ \hline [\![\mathcal{S}]\!] \Rightarrow \#o \geq \ell \end{array}$$

Proof: The proof is by contradiction. Assume that the premises of the rule hold, but not its conclusion. Thus, there is a valuation $\alpha \in [\![\mathcal{S}]\!]$ with $\alpha \models \#o < \ell$, and hence $\alpha \models \#o < \infty$. This means that there is an execution $\xi \in \langle\!\langle \mathcal{S} \rangle\!\rangle$ with

$$\alpha(o) = \xi.\infty(o)$$

With $\nu \stackrel{\mathsf{df}}{=} \#\xi.\infty(o)$ there is an n_1, such that

$$\#\xi.n_1(o) = \nu$$

and an n_2 with

$$\xi.n_2 \models \nu < \ell$$

With $n \stackrel{\text{df}}{=} max(n_1, n_2)$ we have

$$\xi.n \models \#o = \nu \wedge \nu < \ell$$

since o cannot be extended beyond ν in ξ, and because of monotonicity ℓ cannot become smaller, as its arguments are not shortened.

Semantically, the second premise then implies

$$\exists\, m \geq n : \xi.m \models \#o > \nu$$

which contradicts the assumption that $\#o$ does not exceed ν.

Hence the assumption that $\alpha \models \#o < \ell$ is invalid, and for all $\alpha \in [\![\mathcal{S}]\!]$

$$\alpha \models \#o \geq \ell$$

$\square$

In the rule above, o can be replaced by $f(o)$, where f is a continuous function. The rule is also valid if the constant value ∞ is used for ℓ: The component then produces infinite output for any input.

Example

In Section 5.3, the following progress properties of the merge component have been derived:

$$Merge \models \#M_1 Ⓢ o = k \wedge \#i_1 > k \mapsto \#M_1 Ⓢ o > k$$
$$Merge \models \#M_2 Ⓢ o = k \wedge \#i_2 > k \mapsto \#M_2 Ⓢ o > k$$

With the length function $\ell \stackrel{\text{df}}{=} \#i_1$ (and $\ell \stackrel{\text{df}}{=} \#i_2$ for the second input channel), and since Ⓢ is continuous, the rule above allows us to conclude

$$[\![Merge]\!] \Rightarrow \big(\#M_1 Ⓢ o \geq \#i_1\big) \wedge \big(\#M_2 Ⓢ o \geq \#i_2\big)$$

6.4 Methodology

That only length properties are lifted to the black box specification level seems to be quite restrictive. In practice, however, length properties are sufficient for the verification of liveness properties of a state machine's black box view. In Section 2.4 we stated that typical dataflow properties can be formulated as a set of equations, one for each output variable of a component. Each equation can be split into a prefix property (the safety part) and a length property (the liveness part).

The safety part can be verified using the techniques of Section 4; the liveness part can be verified using the techniques of Section 5. For both parts, properties of the black box view can be deduced as shown above.

Example

From Section 6.2, we know the following prefix property of the *Merge* component:

$$[\![Merge]\!] \Rightarrow M_1 \circledS o \sqsubseteq i_1 \wedge M_2 \circledS o \sqsubseteq i_2$$

The following length property of *Merge* has been shown in Section 6.3:

$$[\![Merge]\!] \Rightarrow \big(\#M_1 \circledS o \geq \#i_1\big) \wedge \big(\#M_2 \circledS o \geq \#i_2\big)$$

Together, these properties imply

$$[\![Merge]\!] \Rightarrow M_1 \circledS o = i_1 \wedge M_2 \circledS o = i_2$$

In other words, the state machine from Figure 3 indeed fulfills the black box specification given in Section 2.2.

6.5 Compatibility of the composition operators

The black box composition operator $\otimes$ and the state machine composition operator $||$ are syntactically compatible: They coincide on the definition of the input and output channels of the composed system. Concerning the behavior of the resulting system, the following holds:

$$[\![\mathcal{S}_1 \mid\mid \mathcal{S}_2]\!] \Rightarrow [\![\mathcal{S}_1]\!] \otimes [\![\mathcal{S}_2]\!]$$

The implication is easily proved:

Proof: For the implication, we need to show that when $\alpha \in [\![\mathcal{S}_1 \mid\mid \mathcal{S}_2]\!]$, then also $\alpha \in [\![\mathcal{S}_1]\!]$ and $\alpha \in [\![\mathcal{S}_2]\!]$. Given the left hand side, we know that there exists an execution $\xi \in \langle\!\langle \mathcal{S}_1 \mid\mid \mathcal{S}_2 \rangle\!\rangle$ with

$$\forall\, v \in I \cup O \bullet \alpha(v) = \xi.\infty(v)$$

where $I = (I_1 \cup I_2) \setminus (O_1 \cup O_2)$ and $O = O_1 \cup O_2$.

From Section 3 we know that ξ is also a run of $\mathcal{S}_1$. Since $I_1 \subseteq I \cup O$ and $O_1 \subseteq I \cup O$ we can conclude that

$$\alpha \in [\![\mathcal{S}_1]\!]$$

Similarly, we get $\alpha \in [\![\mathcal{S}_2]\!]$. $\square$

A counterexample shows that the opposite direction does not hold:

Proof: Assume a system $\mathcal{S}_1$ that reads from channel y, and writes on channel x. Both channels can only transmit the message a. The system has only one state and one transition:

$$y?a \rhd x!a$$

Obviously, if the system is fed with a^∞ as input, it reacts by sending a^∞. System $\mathcal{S}_2$ is similar, but it reads from channel x, writes onto y.

The composed system $\mathcal{S}_1 \otimes \mathcal{S}_2$ has no input channels and two output channels. Assigning a^∞ to both x and y represents a possible behavior of this system.

However, this is not a behavior of $\mathcal{S}_1 \mid\mid \mathcal{S}_2$. Here both machines wait for a first message of the other machine; they never send output a message and the only behavior is the one that assigns $\langle\,\rangle$ to both x and y. $\square$

Black box views are an abstraction of a system's behavior. In this abstraction operational information, such as the causality between input and output messages, is lost [4]. One approach to include the causality information also in the black box views is to explicitly introduce time into the communication histories [7, 20]; however, this makes the black box specifications more complex.

7 Example: Communication System

Figure 9 shows a communication system (originally proposed by the VSE group in the DFKI, Saarbrücken, [22]). The system consists of a sender and a receiver connected via a queue component. The queue's buffer can hold N data elements. To ensure that the buffer does not overflow a handshaking protocol is used. We assume that the sender "pushes" data (it sends a datum, then waits for an acknowledgment from the queue), while the receiver "pulls" data (it sends a request to the queue, then awaits a datum). Request and acknowledgment signals are modeled with the singleton set $\mathsf{Signal} = \{\circledast\}$.

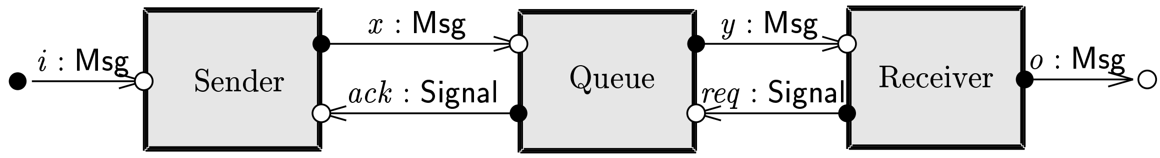

Figure 9: Bounded Buffer

This section first gives black box specifications (Section 7.1) and state machine specifications (Section 7.2) for the communication system's components. Section 7.3 proofs that the state machines imply the safety part of the black box specifications; Section 7.4 shows the same for the liveness part. A discussion about the verification techniques is in Section 7.5.

7.1 Black Box Specifications

The specification of the three components are divided into prefix (safety) and length (progress) properties. The prefix parts simply state the obvious requirement that each component's output is a prefix of its data input.

Sender

in i : Msg, ack : Signal
out x : Msg

$x \sqsubseteq i$
$\#x \geq min(\#i, 1 + \#ack)$

The length property of the sender expresses its "push" behavior: The length of the output is one more than the number of acknowledgments received from the queue, provided there is still data from the environment available.

Receiver

in y : Msg
out req : Signal, o : Signal

$$o \sqsubseteq y$$
$$\#o \geq \#y$$
$$\#req = 1 + \#y$$

The receiver's length property expresses its "pull" behavior: It sends requests initially and after receiving each message from the queue.

Note that here the length property for the requests is an equality. This is because it also incorporates the safety property that the length of *req* must be less or equal than $1 + \#y$; since it is only the number of requests that is relevant, instead of a prefix property a numerical inequality is used as an upper bound on the length of the communication history.

Queue(N)

in x : Msg, req : Signal
out ack : Signal, y : Msg

$$y \sqsubseteq x$$
$$\#y \geq min(\#x, \#req)$$
$$\#ack = min(\#x, \#req + N - 1)$$

The specification for the composition of sender, queue and receiver in our example is shown below.

System(N)

in i : Msg
out o : Signal, x : Msg, ack : Signal, y : Msg, req : Signal

$$x \sqsubseteq i$$
$$y \sqsubseteq x$$
$$o \sqsubseteq y$$

$$\#x \geq min(\#i, 1 + \#ack)$$

$$\#y \geq min(\#x, \#req)$$
$$\#ack = min(\#x, \#req + N - 1)$$

$$\#o \geq \#y$$
$$\#req = 1 + \#y$$

From the specification of *System*(N) above, we can immediately see that the output is a prefix of the input. By some case analysis it can also be shown that the length of the output equals the length of the input. This implies

$$o = i$$

for all input streams i. The communication system implements the identity relation.

7.2 State Machine Specifications

Figure 10 shows the state transitions diagrams of the sender, queue and receiver components. The queue component has an attribute variable q, which holds a finite sequence of messages.

Following Section 3.3, the diagrams can be converted schematically into state transition systems. Below is the STS for each component. For brevity, the names of the STS components and transitions are not differentiated. In the proofs of the verification conditions, it will be clear from the context, which component is referred to.

Sender STS

The STS for the sender is formally defined by

$$\begin{array}{lcl} I & \stackrel{\text{df}}{=} & \{i, ack\} \\ O & \stackrel{\text{df}}{=} & \{x\} \\ V & \stackrel{\text{df}}{=} & \{i, i^\circ, ack, ack^\circ, x, \sigma\} \\ \mathcal{I} & \stackrel{\text{df}}{=} & \sigma = \mathit{Transmit} \wedge i^\circ = \langle\,\rangle \wedge ack^\circ = \langle\,\rangle \wedge x = \langle\,\rangle \\ \mathcal{T} & \stackrel{\text{df}}{=} & \{\tau_1, \tau_2\} \end{array}$$

The transitions τ_1 and τ_2 are the following assertions; they correspond to the arrows in the sender's STD (Figure 10).

$$\begin{array}{lll} \tau_1 \stackrel{\text{df}}{=} \exists\, d. & \sigma = \mathit{Transmit} & \text{We move from the source state} \\ \wedge & \sigma' = \mathit{WaitAck} & \text{to the target state.} \\ \wedge & ft.i^+ = d & \text{There is a message } d \text{ available in channel } i \\ \wedge & i^{\circ\prime} = i^\circ \frown \langle d \rangle & \text{that we consume} \\ \wedge & x' = x \frown \langle d \rangle & \text{and send on channel } x, \\ \wedge & ack^{\circ\prime} = ack^\circ & \text{while we don't read from channel } ack. \\ \wedge & i \sqsubseteq i' \wedge ack \sqsubseteq ack' & \text{The input channels can be extended.} \end{array}$$

$$\begin{array}{ll} \tau_2 \stackrel{\text{df}}{=} & \sigma = \mathit{WaitAck} \\ \wedge & \sigma' = \mathit{Transmit} \\ \wedge & ft.ack^+ = \circledast \\ \wedge & ack^{\circ\prime} = ack^\circ \frown \langle \circledast \rangle \\ \wedge & i^{\circ\prime} = i^\circ \\ \wedge & x' = x \\ \wedge & i \sqsubseteq i' \wedge ack \sqsubseteq ack' \end{array}$$

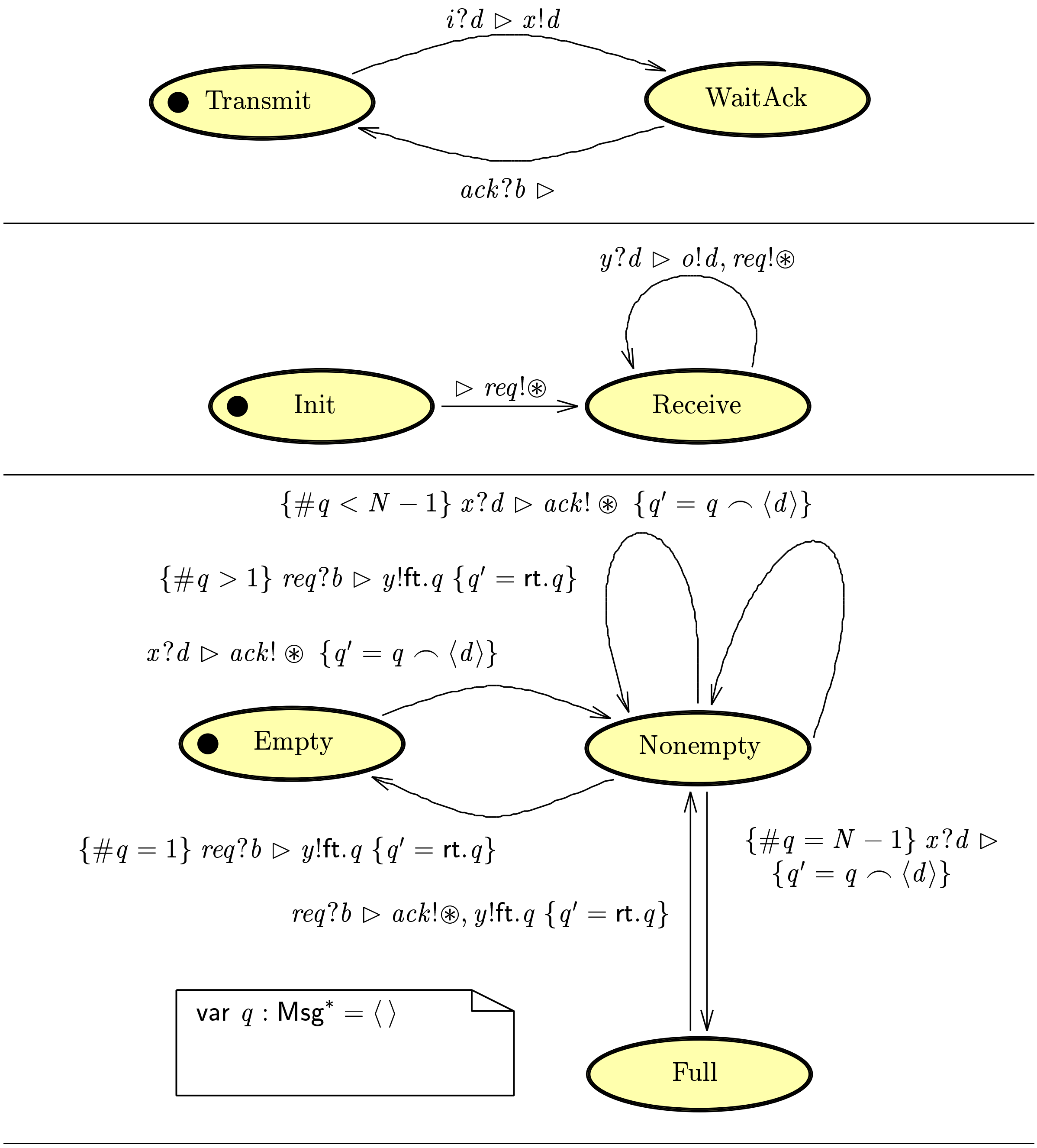

Figure 10: Sender, Receiver and Queue STDs

The environment transition τ^ϵ is defined according to the schema in Section 3.2:

$$\begin{array}{lll} \tau^\epsilon \stackrel{\text{df}}{=} & & \sigma = \sigma' \\ & \wedge & i^{\circ\prime} = i^\circ \\ & \wedge & ack^{\circ\prime} = ack^\circ \\ & \wedge & x' = x \\ & \wedge & i \sqsubseteq i' \wedge ack \sqsubseteq ack' \end{array}$$

Queue STS

The Queue STS contains the sets

$$\begin{array}{lcl} I & \stackrel{\text{df}}{=} & \{x, req\} \\ O & \stackrel{\text{df}}{=} & \{y, ack\} \\ V & \stackrel{\text{df}}{=} & \{x, x^\circ, req, req^\circ, y, ack, q, \sigma\} \\ \mathcal{I} & \stackrel{\text{df}}{=} & \sigma = Empty \wedge x^\circ = \langle\,\rangle \wedge req^\circ = \langle\,\rangle \wedge y = \langle\,\rangle \wedge ack = \langle\,\rangle \wedge q = \langle\,\rangle \\ \mathcal{T} & \stackrel{\text{df}}{=} & \{\tau_1, \tau_2, \tau_3, \tau_4, \tau_5, \tau_6\} \end{array}$$

where the transitions are defined by

$$\begin{array}{lll} \tau_1 \stackrel{\text{df}}{=} \exists\, d. & & \sigma = Empty \\ & \wedge & \sigma' = Nonempty \\ & \wedge & ft.x^+ = d \\ & \wedge & x^{\circ\prime} = x^\circ \frown \langle d \rangle \\ & \wedge & req^{\circ\prime} = req^\circ \\ & \wedge & y' = y \\ & \wedge & ack' = ack \frown \langle \circledast \rangle \\ & \wedge & q' = q \frown \langle d \rangle \\ & \wedge & x \sqsubseteq x' \wedge req \sqsubseteq req' \end{array}$$

$$\begin{array}{lll} \tau_2 \stackrel{\text{df}}{=} & & \sigma = Nonempty \\ & \wedge & \sigma' = Nonempty \\ & \wedge & \#q > 1 \\ & \wedge & ft.req^+ = \circledast \\ & \wedge & req^{\circ\prime} = req^\circ \frown \langle \circledast \rangle \\ & \wedge & x^{\circ\prime} = x^\circ \\ & \wedge & y' = y \frown \langle ft.q \rangle \\ & \wedge & ack' = ack \\ & \wedge & q' = rt.q \\ & \wedge & x \sqsubseteq x' \wedge req \sqsubseteq req' \end{array}$$

$$\begin{array}{lll} \tau_3 \stackrel{\text{df}}{=} \exists\, d. & & \sigma = Nonempty \\ & \wedge & \sigma' = Nonempty \\ & \wedge & \#q < N - 1 \\ & \wedge & ft.x^+ = d \\ & \wedge & x^{\circ\prime} = x^\circ \frown \langle d \rangle \\ & \wedge & req^{\circ\prime} = req^\circ \\ & \wedge & y' = y \\ & \wedge & ack' = ack \frown \langle \circledast \rangle \\ & \wedge & q' = q \frown \langle d \rangle \\ & \wedge & x \sqsubseteq x' \wedge req \sqsubseteq req' \end{array}$$

$$\begin{array}{lll} \tau_4 \stackrel{\text{df}}{=} & & \sigma = Full \\ & \wedge & \sigma' = Nonempty \\ & \wedge & ft.req^+ = \circledast \\ & \wedge & req^{\circ\prime} = req^\circ \frown \langle \circledast \rangle \\ & \wedge & x^{\circ\prime} = x^\circ \\ & \wedge & y' = y \frown \langle ft.q \rangle \\ & \wedge & ack' = ack \frown \langle \circledast \rangle \\ & \wedge & q' = rt.q \\ & \wedge & x \sqsubseteq x' \wedge req \sqsubseteq req' \end{array}$$

$$\tau_5 \stackrel{\text{df}}{=} \exists\, d. \quad \begin{array}{ll} & \sigma = \mathit{Nonempty} \\ \wedge & \sigma' = \mathit{Full} \\ \wedge & \#q = N - 1 \\ \wedge & \mathit{ft}.x^{+} = d \\ \wedge & x^{\circ\prime} = x^{\circ} \frown \langle d \rangle \\ \wedge & \mathit{req}^{\circ\prime} = \mathit{req}^{\circ} \\ \wedge & y' = y \\ \wedge & \mathit{ack}' = \mathit{ack} \\ \wedge & q' = q \frown \langle d \rangle \\ \wedge & x \sqsubseteq x' \wedge \mathit{req} \sqsubseteq \mathit{req}' \end{array} \qquad \tau_6 \stackrel{\text{df}}{=} \begin{array}{ll} & \sigma = \mathit{Nonempty} \\ \wedge & \sigma' = \mathit{Empty} \\ \wedge & \#q = 1 \\ \wedge & \mathit{ft}.\mathit{req}^{+} = \circledast \\ \wedge & \mathit{req}^{\circ\prime} = \mathit{req}^{\circ} \frown \langle \circledast \rangle \\ \wedge & x^{\circ\prime} = x^{\circ} \\ \wedge & y' = y \frown \langle \mathit{ft}.q \rangle \\ \wedge & \mathit{ack}' = \mathit{ack} \\ \wedge & q' = \mathit{rt}.q \\ \wedge & x \sqsubseteq x' \wedge \mathit{req} \sqsubseteq \mathit{req}' \end{array}$$

The environment transition τ^{ϵ} is defined as follows:

$$\tau^{\epsilon} \stackrel{\text{df}}{=} \begin{array}{ll} & \sigma = \sigma' \\ \wedge & x^{\circ\prime} = x^{\circ} \\ \wedge & \mathit{req}^{\circ\prime} = \mathit{req}^{\circ} \\ \wedge & q' = q \\ \wedge & y' = y \\ \wedge & \mathit{ack}' = \mathit{ack} \\ \wedge & x \sqsubseteq x' \\ \wedge & \mathit{req} \sqsubseteq \mathit{req}' \end{array}$$

Receiver STS

The receiver is defined formally through

$$\begin{array}{lcl} I & \stackrel{\text{df}}{=} & \{y\} \\ O & \stackrel{\text{df}}{=} & \{\mathit{req}, o\} \\ V & \stackrel{\text{df}}{=} & \{y, y^{\circ}, \mathit{req}, o, \sigma\} \\ \mathcal{I} & \stackrel{\text{df}}{=} & \sigma = \mathit{Init} \wedge y^{\circ} = \langle\,\rangle \wedge \mathit{req} = \langle\,\rangle \wedge o = \langle\,\rangle \\ \mathcal{T} & \stackrel{\text{df}}{=} & \{\tau_1, \tau_2\} \end{array}$$

with just the following two transitions:

$$\tau_1 \stackrel{\text{df}}{=} \begin{array}{ll} & \sigma = \mathit{Init} \\ \wedge & \sigma' = \mathit{Receive} \\ \wedge & y^{\circ\prime} = y^{\circ} \\ \wedge & \mathit{req}' = \mathit{req} \frown \langle \circledast \rangle \\ \wedge & o' = o \\ \wedge & y \sqsubseteq y' \end{array} \qquad \tau_2 \stackrel{\text{df}}{=} \exists\, d. \quad \begin{array}{ll} & \sigma = \mathit{Receive} \\ \wedge & \sigma' = \mathit{Receive} \\ \wedge & \mathit{ft}.y^{+} = d \\ \wedge & y^{\circ\prime} = y^{\circ} \frown \langle d \rangle \\ \wedge & o' = o \frown \langle d \rangle \\ \wedge & \mathit{req}' = \mathit{req} \frown \langle \circledast \rangle \\ \wedge & y \sqsubseteq y' \end{array}$$

Again, the environment transition τ^ϵ is defined schematically:

$$\begin{aligned}
\tau^\epsilon \stackrel{\mathsf{df}}{=} \quad & \sigma = \sigma' \\
\wedge \quad & y^{\circ\prime} = y^\circ \\
\wedge \quad & req' = req \\
\wedge \quad & o' = o \\
\wedge \quad & y \sqsubseteq y'
\end{aligned}$$

7.3 Safety Proofs

In this section we show that for each of the system's three components, the state machine specification implies the safety part of the black box specification.

For all components, the proof is structured identically:

1. Show that the data output of a component equals the processed part of its input;
2. Conclude that the output is a prefix of the input;
3. Conclude that this also holds for the black box view.

Sender

We show the following property:

$$\mathit{Sender} \models \mathbf{inv}\, x = i^\circ$$

According to the rules in sections 4.1 and 4.2 we need to prove

$$\mathcal{I} \;\Rightarrow\; x = i^\circ \tag{5}$$

$$\bigwedge_{\tau\in\mathcal{T}} \tau \wedge (x = i^\circ) \;\Rightarrow\; x' = i^{\circ\prime} \tag{6}$$

Since $\mathcal{I} \Rightarrow x = \langle\,\rangle = i^\circ$, obligation (5) is trivially fulfilled. We now show (6) for all τ:

- Transition τ_1:

$$\begin{aligned}
& x = i^\circ \wedge \tau_1 \\
& \quad \Rightarrow x \frown \langle d\rangle = i^\circ \frown \langle d\rangle \\
& \quad \Rightarrow x' = i^{\circ\prime}
\end{aligned}$$

- Transition τ_2:

$$\begin{aligned}
& x = i^\circ \wedge \tau_2 \\
& \quad \Rightarrow x' = i^{\circ\prime}
\end{aligned}$$

- Transition τ^ϵ:

$$\begin{aligned}&x = i^\circ \wedge \tau_2\\&\qquad \Rightarrow x' = i^{\circ\prime}\end{aligned}$$

Since $i^\circ \sqsubseteq i$ is also an invariant of the sender, we can conclude that

$$\mathit{Sender} \models \mathbf{inv}\, x \sqsubseteq i$$

and therefore

$$[\![\mathit{Sender}]\!] \Rightarrow x \sqsubseteq i$$

Thus, the state machine of the sender implies the safety part of the sender's black box specification.

Queue

For the queue component, we show the following property:

$$\mathit{Queue} \models \mathbf{inv}\, y \frown q = x^\circ$$

Since $\mathcal{I} \Rightarrow y = q = x^\circ = \langle\,\rangle$ the property above holds initially. We now show that is also stable, and therefore indeed an invariant:

- Transition τ_1:

  $$\begin{aligned}&y \frown q = x^\circ \wedge \tau_1\\&\qquad \Rightarrow y \frown q \frown \langle \mathsf{ft}.x^+\rangle = x^\circ \frown \langle \mathsf{ft}.x^+\rangle\\&\qquad \Rightarrow y' \frown q' = x^{\circ\prime}\end{aligned}$$

  The proof is analogous for the transitions τ_3 and τ_5.

- Transition τ_2:

  $$\begin{aligned}&y \frown q = x^\circ \wedge \tau_2\\&\qquad \Rightarrow y \frown \langle \mathit{ft}.q\rangle \frown \mathit{rt}.q = x^\circ\\&\qquad \Rightarrow y' \frown q' = x^{\circ\prime}\end{aligned}$$

  The proof is analogous for the transitions τ_4 and τ_6.

- Transition τ^ϵ:

  $$\begin{aligned}&y \frown q = x^\circ \wedge \tau^\epsilon\\&\qquad \Rightarrow y \frown q = x^\circ \wedge y' = y \wedge x^{\circ\prime} = x^\circ \wedge q' = q\\&\qquad \Rightarrow y' \frown q' = x^{\circ\prime}\end{aligned}$$

From the invariants $y \frown q = x^\circ$ and the invariant $x^\circ \sqsubseteq x$, we conclude

$$Queue \models \mathbf{inv}\; y \sqsubseteq x$$

Hence, the queue component fulfills the safety part of its black box specification, since

$$[\![Queue]\!] \Rightarrow y \sqsubseteq x$$

Receiver

For the receiver, we show

$$Receiver \models \mathbf{inv}\; o = y^\circ$$

That this property holds initially is immediate since

$$\mathcal{I} \Rightarrow o = y^\circ = \langle\,\rangle$$

It remains to show that the property is stable under the two receiver transitions τ_1 and τ_2 and the environment transition τ^ϵ.

- Transition τ_1 (the proof is analogous for τ^ϵ):

$$\begin{aligned} & o = y^\circ \wedge \tau_1 \\ & \quad \Rightarrow o' = y^{\circ\prime} \end{aligned}$$

- Transition τ_2:

$$\begin{aligned} & o = y^\circ \wedge \tau_2 \\ & \quad o \frown \langle \mathsf{ft}.y^+ \rangle = y^\circ \frown \langle \mathsf{ft}.y^+ \rangle \\ & \quad \Rightarrow o' = y^{\circ\prime} \end{aligned}$$

Thus, $o = y^\circ$ is an invariant of the receiver. Since also $i^\circ \sqsubseteq y$ is an invariant, we can conclude

$$Receiver \models \mathbf{inv}\; o \sqsubseteq y$$

From this, we can immediately conclude the safety part of the receiver's black box specification:

$$[\![Receiver]\!] \Rightarrow o \sqsubseteq y$$

7.4 Liveness Proofs

For each component, the liveness part of the black box specification is derived from the component's output extension properties.

Usually, the liveness proofs require some knowledge about the relation between control state, attribute values and the length or contents of the variables i°. Such relations are expressed by additional invariants of the components. For the liveness proofs below, we just list the invariants. Their proof is analogous to the proof of the prefix properties in the previous section.

Sender

We need to prove that the output $x \in O$ is extended; the length function ℓ is the *min*-Term of the black box specification: $\ell \stackrel{\mathsf{df}}{=} \mathit{min}(\#i, 1 + \#ack)$.

First we prove the following two properties that reflect the effect of the sender's two transitions.

- Transition τ_1 indeed extends the output:

$$(1)\ \sigma = \mathit{Transmit} \wedge \#x = k \wedge k < \ell \mapsto \#x > k$$

- Transition τ_2, however, leaves the output unchanged. The length expression may become larger, but in any case it will stay larger than k:

$$(2)\ \sigma = \mathit{WaitAck} \wedge \#x = k \wedge k < \ell \mapsto \sigma = \mathit{Transmit} \wedge \#x = k \wedge k < \ell$$

The first property is proven with the output extension rule. We need to show the following premises, where we choose τ_1 as the helpful transition:

$$(1.1)\ \sigma = \mathit{Transmit} \wedge \#x = k \wedge k < \ell \Rightarrow \mathsf{En}(\tau_1)$$

$$(1.2)\ \sigma = \mathit{Transmit} \wedge \#x = k \wedge k < \ell \wedge \tau_1 \Rightarrow \#x' > k$$

Premise (1.1) is fulfilled, since the enabledness condition of τ_1 corresponds to

$$\sigma = \mathit{Transmit} \wedge \#i^+ > 0$$

which holds since

$$\#i \geq \mathit{min}(\#i, 1 + \#ack) = \ell > k = \#x = \#i^\circ$$

using the fact that $\#x = \#i^\circ$ as shown in Section 7.3. Hence, $\#i^+ = \#i - \#i^\circ > 0$.

Premise (1.2) follows immediately from the definition of τ_1 with $x' = x \frown \langle d \rangle$.

For property (2), we use the ensure rule; the helpful transition in this case is τ_2. We need to discharge the following three premises:

$$\begin{aligned}
(2.1)\ \mathcal{S} \models & \big(\sigma = \mathit{WaitAck} \wedge \#x = k \wedge \ell > k\big) \wedge \neg\big(\sigma = \mathit{Transmit} \wedge \#x = k \wedge \ell > k\big) \\
& \qquad\qquad\qquad\qquad \mathbf{co} \\
& \big(\sigma = \mathit{WaitAck} \wedge \#x = k \wedge \ell > k\big) \vee \big(\sigma = \mathit{Transmit} \wedge \#x = k \wedge \ell > k\big)
\end{aligned}$$

$$\begin{aligned}
(2.2)\ & \big(\sigma = \mathit{WaitAck} \wedge \#x = k \wedge \ell > k\big) \wedge \\
& \qquad \neg\big(\sigma = \mathit{Transmit} \wedge \#x = k \wedge \ell > k\big) \Rightarrow \mathsf{En}(\tau_2)
\end{aligned}$$

$$\begin{aligned}
(2.3)\ & \big(\sigma = \mathit{WaitAck} \wedge \#x = k \wedge \ell > k\big) \wedge \neg\big(\sigma = \mathit{Transmit} \wedge \#x = k \wedge \ell > k\big) \wedge \\
& \qquad \tau_2 \Rightarrow \sigma' = \mathit{Transmit} \wedge \#x' = k \wedge \ell' > k
\end{aligned}$$

Premise (2.1) holds, since transition τ_1 is not enabled in states that satisfy the premise's left hand side; the environment transition leaves σ as well as $\#x$ unchanged, while ℓ cannot become smaller. Finally, transition τ_2 leads to a state where $\sigma = \mathit{Transmit} \wedge \#x = k \wedge \ell > k$. This also implies premise (2.3).

For premise (2.2), we need to show that $\#ack^+ > 0$. This premise requires an additional invariant, namely

$$\mathcal{S} \models \mathbf{inv}(\sigma = \mathit{Transmit} \Rightarrow \#x = \#ack^\circ) \wedge (\sigma = \mathit{WaitAck} \Rightarrow \#x = 1 + \#ack^\circ)$$

The proof of this invariant follows the structure of the proofs in Section 7.3.

From this invariant and the left hand side of the implication (2.2), we conclude

$$1 + \#ack \geq \ell > k = \#x = 1 + \#ack^\circ$$

Hence, $\#ack^+ = \#ack - \#ack^\circ > 0$. □

The two leadsto properties (1) and (2) can be combined by the transitivity rule, which yields:

$$(3)\ \sigma = \mathit{WaitAck} \wedge \#x = k \wedge \ell > k \mapsto \#x > k$$

Properties (1) and (3) are combined by the disjunction rule:

$$(4)\ (\sigma = \mathit{WaitAck} \vee \sigma = \mathit{Transmit}) \wedge \#x = k \wedge \ell > k \mapsto \#x > k$$

Since the two control states *WaitAck* and *Transmit* are the only control states of the sender, the disjunction on the left hand side of (4) is equivalent to true; thus, (4) can be simplified which yields

$$(5)\ \#x = k \wedge \ell > k \mapsto \#x > k$$

Now, from (5) we obtain

$$[\![\mathit{Sender}]\!] \Rightarrow \#x \geq \mathit{min}(\#i, 1 + \#ack)$$

$\square$

Queue

The queue has two output variables. For each output, the following extension properties are valid:

$$\begin{aligned}
&\#y = k \wedge min(\#x, \#req) > k && \mapsto \#y > k \\
&\#ack = k \wedge min(\#x, \#req + N) > k && \mapsto \#ack > k
\end{aligned}$$

We want to show only the first property here.

To prove the first property, we need the following invariants which relate control and data state, as well as control state and the lengths of the processed input variables. These invariants can be shown in the same style as the prefix property in Section 7.3.

$$\begin{aligned}
Queue \models \mathbf{inv}\ &\big(\sigma = Empty \wedge \#q = 0\big) \vee \\
&\big(\sigma = Nonempty \wedge 1 \leq \#q \leq N - 1\big) \vee \\
&\big(\sigma = Full \wedge \#q = N\big)
\end{aligned}$$

$$\begin{aligned}
Queue \models \mathbf{inv}\ &\big(\sigma = Empty \Rightarrow \#y = \#req^{\circ} \wedge \#ack = \#x^{\circ}\big) \wedge \\
&\big(\sigma = Nonempty \Rightarrow \#y = \#req^{\circ} \wedge \#ack = \#x^{\circ}\big) \wedge \\
&\big(\sigma = Full \Rightarrow \#y = \#req^{\circ} \wedge \#ack + 1 = \#x^{\circ}\big)
\end{aligned}$$

$$Queue \models \mathbf{inv}\ \#x^{\circ} = \#req^{\circ} + \#q$$

The transitions that extend y are τ_2, τ_4, τ_6. Choosing these transitions as helpful transitions in the output extension rule, we can show (with $\ell \stackrel{\text{df}}{=} min(\#x, \#req)$):

$$\begin{aligned}
&(1)\ \sigma = NonEmpty \wedge \#q > 1 \wedge \#y = k \wedge \ell > k && \mapsto \#y > k \\
&(2)\ \sigma = Full \wedge \#q = N \wedge \#y = k \wedge \ell > k && \mapsto \#y > k \\
&(3)\ \sigma = NonEmpty \wedge \#q = 1 \wedge \#y = k \wedge \ell > k && \mapsto \#y > k
\end{aligned}$$

In each rule application, the invariants above has to be used to show that the transitions τ_2, τ_4, τ_6, respectively, are enabled.

Examining the first two invariants above, we note that the only state where no helpful transition is enabled when $\ell > k$ is the control state *Empty*.

Now, because of the invariants, we know that when $\sigma = Empty$, then also $\#q = 0$, and

$$\#x^{\circ} = \#req^{\circ} = \#y = k < \ell \leq \#x$$

This means that transition τ_1 is enabled, since $\#x^+ > 0$. Therefore, with

$$\begin{array}{lcl} \Phi & \equiv & \sigma = \mathit{Empty} \wedge \#q = 0 \wedge \#y = k \wedge \ell > k \\ \Psi & \equiv & \sigma = \mathit{NonEmpty} \wedge \#q = 1 \wedge \#y = k \wedge \ell > k \end{array}$$

we got

$$\Phi \wedge \neg\, \Psi \Rightarrow \mathsf{En}(\tau_1)$$

Since τ_1 leads to the state *NonEmpty* and increases the length of q to 1, without changing y and not decreasing ℓ, we also have

$$\Phi \wedge \neg\, \Psi \wedge \tau_1 \Rightarrow \Psi'$$

The property

$$\Phi \wedge \neg\, \Psi \ \mathbf{co}\ \Phi \vee \Psi$$

holds for τ_1, as already seen. The other transition in $\mathcal{T}$ are not enabled, and for τ^ϵ the validity of Φ does not change. So, we can use the ensure rule and conclude

$$\begin{array}{l} (4)\ \sigma = \mathit{Empty} \wedge \#q = 0 \wedge \#y = k \wedge \ell > k \\ \qquad\qquad\qquad \mapsto \sigma = \mathit{NonEmpty} \wedge \#q = 1 \wedge \#y = k \wedge \ell > k \end{array}$$

By transitivity of $\mapsto$, we obtain from (4) and (3):

$$(5)\ \sigma = \mathit{Empty} \wedge \#q = 0 \wedge \#y = k \wedge \ell > k \mapsto \#y > k$$

The properties (1), (2), (3), (5) can be combined with a finite variant of the disjunction rule; after invariant elimination on the left hand side, we obtain

$$\#y = k \wedge \mathit{min}(\#x, \#\mathit{req}) > k \mapsto \#y > k$$

□

Receiver

The liveness proof of the receiver is quite similar to the one for the sender. We omit the proof here. For each output of the receiver, it needs two applications of the output extension rule, one application of the transitivity rule and one application of the disjunction rule.

7.5 Comments

The proofs that the state machines satisfy the black box specifications might seem frighteningly complicated. We believe, however, that this is less a matter of complexity, and more a matter of the total size of the proof. The verification conditions themselves can be reduced to implications in predicate logic, and are not too difficult to discharge.

The deal with the sheer number of verification conditions, obviously some kind of tool support in the form of interactive theorem provers is needed. Since the verification conditions themselves are mainly first-order logic, and no elaborate theory of streams is needed (see [17] for a discussion of the difficulties of stream formalizations), the demands of the prover are not very high.

Another problem is the structure of the proofs. A solution might be the use of verification diagrams [5] which represent proof structures as directed diagrams. The vertices are labeled with state predicates, the labels with transitions. Each transition represents a verification condition.

8 Conclusion

This report shows how to combine state-based and history-based specification and verification of safety and liveness properties of distributed systems. Properties for state machines are formulated in a UNITY-like language; since our approach is based on proof principles for invariants and leadsto properties, other linear-time temporal logics [25, 24] can be used as well.

Dataflow systems are interference free: Components cannot disable transitions of other components. Noninterference means that our proof system is compositional for both safety and liveness properties. This has also been exploited for UNITY by Charpentier and Chandy [16]; however, they do not use their dataflow properties to reason about complete communication histories.

Since the number of verification conditions for concrete systems can be quite large, some kind of tool support is needed. Tool support is not infeasible, since each condition itself is rather simple and can be expressed in first-order logic. In particular, the black box properties that refer to potentially infinite streams are derived from state properties that refer only to finite streams. Hence, the comparatively simple theory of lists is sufficient to discharge the verification conditions; the difficulties of the encoding of infinite streams or lazy lists —which requires corecursion or a CPO theory [17]— can be avoided.

As a case study in tool support, the safety properties of the communication system example have been verified using the STeP [2] proof environment. Except for the instantiation of some stream axioms, the verification conditions are discharged automatically. Using a theorem prover with stronger automatization, such as Isabelle [29], would further reduce the manual effort for discharging the proof obligations. Recently, a Unity formalization in Isabelle has been developed [30], which could probably be adapted to our framework.

Previous work on the combination of state machine descriptions and the stream-based specification of Focus has dealt primarily with stream-based semantics of state machines [11, 19]; the connection of that work to the proof principles in this report and Broy's verification of the Alternating Bit Protocol [6] has to be explored. An open question is also the connection to state machine refinement calculi [32] and refinement in general [9, 10]. For the special case of architectural refinement [31], our proof techniques allow the formulation and verification of invariants.

Our specification and proof techniques are so far only suited for time-independent systems. The extension of history-based specifications raises some interesting questions [8]. A straightforward solution might be to explicitly include "time ticks" in the message streams [18]. Such time ticks can also be used to ensure progress of a state machine. But also without explicit time, progress is not restricted to the weak fairness condition of Section 3.4. An alternative would be to just demand that some transition is taken whenever at least one transition is persistently enabled; the definition of interleaving composition, however, would be slightly more complicated.

Finally, our techniques can be adapted to different state-based description techniques.

SDL [1, 23] and ROOM [33], in particular, would be good candidates for a concrete state machine syntax, given their use in the specification of communication protocols.

Acknowledgments. This report benefited from many stimulating discussions with Manfred Broy. We thank Stephan Merz, Bernhard Rumpe, Robert Sandner and Bernhard Schätz for comments on a draft version of this report.

References

[1] F. Beline, D. Hogrefe, and A. Sarma. *SDL with Applications from Protocol Specification.* Prentice-Hall, 1991.

[2] N. Bjørner, A. Browne, E. Chang, M. Colón, A. Kapur, Z. Manna, H. B. Sipma, and T. E. Uribe. STeP: Deductive-Algorithmic Verification of Reactive and Real-time Systems. In *CAV'96*, volume 1102 of *Lecture Notes in Computer Science*, pages 415–418, 1996.

[3] M. Breitling, U. Hinkel, and K. Spies. Formale Entwicklung verteilter reaktiver Systeme mit Focus. In *Formale Beschreibungstechniken für verteilte Systeme, 8. GI/ITG Fachgespräch*, 1998.

[4] J. D. Brock and W. B. Ackermann. Scenarios: A model of nondeterministic computation. In J. Diaz and I.Ramos, editors, *Lecture Notes in Computer Science 107*, pages 225–259, 1981.

[5] I. A. Browne, Z. Manna, and H. B. Sipma. Generalized temporal verification diagrams. In *Lecture Notes in Computer Science 1026*, pages 484–498, 1995.

[6] M. Broy. From states to histories. To be published.

[7] M. Broy. Applicative real time programming. In *FIP World Congress*, Information Processing 83, pages 259–264, 1983.

[8] M. Broy. Functional specification of time sensitive communicating systems. In J. W. de Bakker, W. P. de Roever, and G. Rozenberg, editors, *Models, Formalism, Correctness. Lecture Notes in Computer Science 430*, pages 153–179. Springer, 1990.

[9] M. Broy. Compositional refinement of interactive systems. Working Material, International Summer School on Program Design Calculi, August 1992.

[10] M. Broy. (Inter-) action refinement: the easy way. Working Material, International Summer School on Program Design Calculi, August 1992.

[11] M. Broy. The Specification of System Components by State Transition Diagrams. Technical Report TUM-I9729, Institut für Informatik, Technische Universität München, 1997.

[12] M. Broy, F. Dederichs, C. Dendorfer, M. Fuchs, T. F. Gritzner, and R. Weber. The Design of Distributed Systems: An Introduction to Focus—Revised Version. Technical Report TUM-I9202-2, Institut für Informatik, Technische Universität München, 1993.

[13] M. Broy, F. Huber, B. Paech, B. Rumpe, and K. Spies. Software and system modeling based on a unified formal semantics. In Manfred Broy and Bernhard Rumpe,

editors, *Requirements Targeting Software and Systems Engineering, International Workshop RTSE'97, LNCS 1526*. Springer, 1998.

[14] M. Broy, F. Huber, and B. Schätz. AutoFocus – ein Werkzeugprototyp zur Entwicklung eingebetteter Systeme. *Informatik Forschung und Entwicklung*, 14(3):121–134, 1999.

[15] M. Broy and K. Stølen. Focus on system development. To be published.

[16] M. Charpentier and K. M. Chandy. Towards a compositional approach to the design and verification of distributed systems. In J.M. Wing, J. Woodcock, and J. Davies, editors, *FM'99 - Formal Methods. Lecture Notes in Computer Science 1708*, pages 570–589. Springer, 1999.

[17] M. Devillers, D. Griffioen, and O. Müller. Possibly infinite sequences in theorem provers: A comparative study. In *TPHOL'97, Proc. of the 10th International Workshop on Theorem Proving in Higher Order Logics, LNCS 1275*, pages 89–104, 1997.

[18] M. Fuchs and K. Stølen. A formal method for hardware/software co-design. Technical Report TUM-I9517, Institut für Informatik, Technische Universität München, 1995.

[19] R. Grosu and B. Rumpe. Concurrent timed port automata. Technical Report TUM-I9533, Institut für Informatik, Technische Universität München, 1995.

[20] E. C. R. Hehner. *A Practical Theory of Programming.* Springer-Hall, 1993.

[21] F. Huber, B. Schätz, A. Schmidt, and K. Spies. Autofocus—a tool for distributed systems specification. In *Proceedings FTRTFT'96 — Formal Techniques in Real-Time and Fault-Tolerant Systems*, LNCS 1135, 1996.

[22] D. Hutter, H. Mantel, G. Rock, and W. Stephan. Nebenläufige verteilte Systeme: Ein Producer/Consumer Beispiel. Internal Manuscript, DFKI, 1998.

[23] ITU-T. *Recommendation Z.100, Specification and Description Language (SDL).* ITU, 1993.

[24] L. Lamport. The temporal logic of actions. *ACM Transactions on Programming Languages*, 6(3):872–923, May 1994.

[25] Z. Manna and A. Pnueli. *The Temporal Logic of Reactive and Concurrent Systems — Specification.* Springer, 1991.

[26] J. Misra. A logic for concurrent programming: Progress. *Journal of Computer and Software Engineering*, 3(2):273–300, 1995.

[27] J. Misra. A logic for concurrent programming: Safety. *Journal of Computer and Software Engineering*, 3(2):239–272, 1995.

[28] L. C. Paulson. *Logic and Computation*. Cambridge University Press, 1987.

[29] L. C. Paulson. *Isabelle: A Generic Theorem Prover*, volume 828 of *Lecture Notes in Computer Science*. Springer, 1994.

[30] L. C. Paulson. Mechanizing UNITY in Isabelle. Technical Report 467, University of Cambridge, Computer Laboratory, 1999.

[31] J. Philipps and B. Rumpe. Refinement of pipe and filter architectures. In J.M. Wing, J. Woodcock, and J. Davies, editors, *FM'99 - Formal Methods. Lecture Notes in Computer Science 1708*, pages 96–115. Springer, 1999.

[32] B. Rumpe. *Formale Methodik des Entwurfs verteilter objektorientierter Systeme.* Herbert Utz Verlag Wissenschaft, 1996. PhD thesis, Technische Universität München.

[33] B. Selic, G. Gullekson, and P. T. Ward. *Real-time object-oriented modeling.* John Wiley & Sons, 1994.

SFB 342: Methoden und Werkzeuge für die Nutzung paralleler Rechnerarchitekturen

bisher erschienen :

Reihe A

Liste aller erschienenen Berichte von 1990-1994
auf besondere Anforderung

342/01/95 A Hans-Joachim Bungartz: Higher Order Finite Elements on Sparse Grids

342/02/95 A Tao Zhang, Seonglim Kang, Lester R. Lipsky: The Performance of Parallel Computers: Order Statistics and Amdahl's Law

342/03/95 A Lester R. Lipsky, Appie van de Liefvoort: Transformation of the Kronecker Product of Identical Servers to a Reduced Product Space

342/04/95 A Pierre Fiorini, Lester R. Lipsky, Wen-Jung Hsin, Appie van de Liefvoort: Auto-Correlation of Lag-k For Customers Departing From Semi-Markov Processes

342/05/95 A Sascha Hilgenfeldt, Robert Balder, Christoph Zenger: Sparse Grids: Applications to Multi-dimensional Schrödinger Problems

342/06/95 A Maximilian Fuchs: Formal Design of a Model-N Counter

342/07/95 A Hans-Joachim Bungartz, Stefan Schulte: Coupled Problems in Microsystem Technology

342/08/95 A Alexander Pfaffinger: Parallel Communication on Workstation Networks with Complex Topologies

342/09/95 A Ketil Stølen: Assumption/Commitment Rules for Data-flow Networks - with an Emphasis on Completeness

342/10/95 A Ketil Stølen, Max Fuchs: A Formal Method for Hardware/Software Co-Design

342/11/95 A Thomas Schnekenburger: The ALDY Load Distribution System

342/12/95 A Javier Esparza, Stefan Römer, Walter Vogler: An Improvement of McMillan's Unfolding Algorithm

342/13/95 A Stephan Melzer, Javier Esparza: Checking System Properties via Integer Programming

342/14/95 A Radu Grosu, Ketil Stølen: A Denotational Model for Mobile Point-to-Point Dataflow Networks

342/15/95 A Andrei Kovalyov, Javier Esparza: A Polynomial Algorithm to Compute the Concurrency Relation of Free-Choice Signal Transition Graphs

342/16/95 A Bernhard Schätz, Katharina Spies: Formale Syntax zur logischen Kernsprache der Focus-Entwicklungsmethodik

342/17/95 A Georg Stellner: Using CoCheck on a Network of Workstations

342/18/95 A Arndt Bode, Thomas Ludwig, Vaidy Sunderam, Roland Wismüller: Workshop on PVM, MPI, Tools and Applications

Reihe A

342/19/95 A	Thomas Schnekenburger: Integration of Load Distribution into ParMod-C
342/20/95 A	Ketil Stølen: Refinement Principles Supporting the Transition from Asynchronous to Synchronous Communication
342/21/95 A	Andreas Listl, Giannis Bozas: Performance Gains Using Subpages for Cache Coherency Control
342/22/95 A	Volker Heun, Ernst W. Mayr: Embedding Graphs with Bounded Treewidth into Optimal Hypercubes
342/23/95 A	Petr Jančar, Javier Esparza: Deciding Finiteness of Petri Nets up to Bisimulation
342/24/95 A	M. Jung, U. Rüde: Implicit Extrapolation Methods for Variable Coefficient Problems
342/01/96 A	Michael Griebel, Tilman Neunhoeffer, Hans Regler: Algebraic Multigrid Methods for the Solution of the Navier-Stokes Equations in Complicated Geometries
342/02/96 A	Thomas Grauschopf, Michael Griebel, Hans Regler: Additive Multilevel-Preconditioners based on Bilinear Interpolation, Matrix Dependent Geometric Coarsening and Algebraic-Multigrid Coarsening for Second Order Elliptic PDEs
342/03/96 A	Volker Heun, Ernst W. Mayr: Optimal Dynamic Edge-Disjoint Embeddings of Complete Binary Trees into Hypercubes
342/04/96 A	Thomas Huckle: Efficient Computation of Sparse Approximate Inverses
342/05/96 A	Thomas Ludwig, Roland Wismüller, Vaidy Sunderam, Arndt Bode: OMIS — On-line Monitoring Interface Specification
342/06/96 A	Ekkart Kindler: A Compositional Partial Order Semantics for Petri Net Components
342/07/96 A	Richard Mayr: Some Results on Basic Parallel Processes
342/08/96 A	Ralph Radermacher, Frank Weimer: INSEL Syntax-Bericht
342/09/96 A	P.P. Spies, C. Eckert, M. Lange, D. Marek, R. Radermacher, F. Weimer, H.-M. Windisch: Sprachkonzepte zur Konstruktion verteilter Systeme
342/10/96 A	Stefan Lamberts, Thomas Ludwig, Christian Röder, Arndt Bode: PFSLib – A File System for Parallel Programming Environments
342/11/96 A	Manfred Broy, Gheorghe Ştefănescu: The Algebra of Stream Processing Functions
342/12/96 A	Javier Esparza: Reachability in Live and Safe Free-Choice Petri Nets is NP-complete
342/13/96 A	Radu Grosu, Ketil Stølen: A Denotational Model for Mobile Many-to-Many Data-flow Networks
342/14/96 A	Giannis Bozas, Michael Jaedicke, Andreas Listl, Bernhard Mitschang, Angelika Reiser, Stephan Zimmermann: On Transforming a Sequential SQL-DBMS into a Parallel One: First Results and Experiences of the MIDAS Project

Reihe A

342/15/96 A Richard Mayr: A Tableau System for Model Checking Petri Nets with a Fragment of the Linear Time μ -Calculus
342/16/96 A Ursula Hinkel, Katharina Spies: Anleitung zur Spezifikation von mobilen, dynamischen Focus-Netzen
342/17/96 A Richard Mayr: Model Checking PA-Processes
342/18/96 A Michaela Huhn, Peter Niebert, Frank Wallner: Put your Model Checker on Diet: Verification on Local States
342/01/97 A Tobias Müller, Stefan Lamberts, Ursula Maier, Georg Stellner: Evaluierung der Leistungsf"ahigkeit eines ATM-Netzes mit parallelen Programmierbibliotheken
342/02/97 A Hans-Joachim Bungartz and Thomas Dornseifer: Sparse Grids: Recent Developments for Elliptic Partial Differential Equations
342/03/97 A Bernhard Mitschang: Technologie f"ur Parallele Datenbanken - Bericht zum Workshop
342/04/97 A nicht erschienen
342/05/97 A Hans-Joachim Bungartz, Ralf Ebner, Stefan Schulte: Hierarchische Basen zur effizienten Kopplung substrukturierter Probleme der Strukturmechanik
342/06/97 A Hans-Joachim Bungartz, Anton Frank, Florian Meier, Tilman Neunhoeffer, Stefan Schulte: Fluid Structure Interaction: 3D Numerical Simulation and Visualization of a Micropump
342/07/97 A Javier Esparza, Stephan Melzer: Model Checking LTL using Constraint Programming
342/08/97 A Niels Reimer: Untersuchung von Strategien für verteiltes Last- und Ressourcenmanagement
342/09/97 A Markus Pizka: Design and Implementation of the GNU INSEL-Compiler gic
342/10/97 A Manfred Broy, Franz Regensburger, Bernhard Schätz, Katharina Spies: The Steamboiler Specification - A Case Study in Focus
342/11/97 A Christine Röckl: How to Make Substitution Preserve Strong Bisimilarity
342/12/97 A Christian B. Czech: Architektur und Konzept des Dycos-Kerns
342/13/97 A Jan Philipps, Alexander Schmidt: Traffic Flow by Data Flow
342/14/97 A Norbert Fröhlich, Rolf Schlagenhaft, Josef Fleischmann: Partitioning VLSI-Circuits for Parallel Simulation on Transistor Level
342/15/97 A Frank Weimer: DaViT: Ein System zur interaktiven Ausführung und zur Visualisierung von INSEL-Programmen
342/16/97 A Niels Reimer, Jürgen Rudolph, Katharina Spies: Von FOCUS nach INSEL - Eine Aufzugssteuerung
342/17/97 A Radu Grosu, Ketil Stølen, Manfred Broy: A Denotational Model for Mobile Point-to-Point Data-flow Networks with Channel Sharing
342/18/97 A Christian Röder, Georg Stellner: Design of Load Management for Parallel Applications in Networks of Heterogenous Workstations

Reihe A

342/19/97 A	Frank Wallner: Model Checking LTL Using Net Unfoldings
342/20/97 A	Andreas Wolf, Andreas Kmoch: Einsatz eines automatischen Theorembeweisers in einer taktikgesteuerten Beweisumgebung zur Lösung eines Beispiels aus der Hardware-Verifikation – Fallstudie –
342/21/97 A	Andreas Wolf, Marc Fuchs: Cooperative Parallel Automated Theorem Proving
342/22/97 A	T. Ludwig, R. Wismüller, V. Sunderam, A. Bode: OMIS - On-line Monitoring Interface Specification (Version 2.0)
342/23/97 A	Stephan Merkel: Verification of Fault Tolerant Algorithms Using PEP
342/24/97 A	Manfred Broy, Max Breitling, Bernhard Schätz, Katharina Spies: Summary of Case Studies in Focus - Part II
342/25/97 A	Michael Jaedicke, Bernhard Mitschang: A Framework for Parallel Processing of Aggregat and Scalar Functions in Object-Relational DBMS
342/26/97 A	Marc Fuchs: Similarity-Based Lemma Generation with Lemma-Delaying Tableau Enumeration
342/27/97 A	Max Breitling: Formalizing and Verifying TimeWarp with FOCUS
342/28/97 A	Peter Jakobi, Andreas Wolf: DBFW: A Simple DataBase FrameWork for the Evaluation and Maintenance of Automated Theorem Prover Data (incl. Documentation)
342/29/97 A	Radu Grosu, Ketil Stølen: Compositional Specification of Mobile Systems
342/01/98 A	A. Bode, A. Ganz, C. Gold, S. Petri, N. Reimer, B. Schiemann, T. Schnekenburger (Herausgeber): "'Anwendungsbezogene Lastverteilung"', ALV'98
342/02/98 A	Ursula Hinkel: Home Shopping - Die Spezifikation einer Kommunikationsanwendung in Focus
342/03/98 A	Katharina Spies: Eine Methode zur formalen Modellierung von Betriebssystemkonzepten
342/04/98 A	Stefan Bischof, Ernst-W. Mayr: On-Line Scheduling of Parallel Jobs with Runtime Restrictions
342/05/98 A	St. Bischof, R. Ebner, Th. Erlebach: Load Balancing for Problems with Good Bisectors and Applications in Finite Element Simulations: Worst-case Analysis and Practical Results
342/06/98 A	Giannis Bozas, Susanne Kober: Logging and Crash Recovery in Shared-Disk Database Systems
342/07/98 A	Markus Pizka: Distributed Virtual Address Space Management in the MoDiS-OS
342/08/98 A	Niels Reimer: Strategien für ein verteiltes Last- und Ressourcenmanagement
342/09/98 A	Javier Esparza, Editor: Proceedings of INFINITY'98
342/10/98 A	Richard Mayr: Lossy Counter Machines
342/11/98 A	Thomas Huckle: Matrix Multilevel Methods and Preconditioning

Reihe A

342/12/98 A Thomas Huckle: Approximate Sparsity Patterns for the Inverse of a Matrix and Preconditioning

342/13/98 A Antonin Kucera, Richard Mayr: Weak Bisimilarity with Infinite-State Systems can be Decided in Polynomial Time

342/01/99 A Antonin Kucera, Richard Mayr: Simulation Preorder on Simple Process Algebras

342/02/99 A Johann Schumann, Max Breitling: Formalisierung und Beweis einer Verfeinerung aus FOCUS mit automatischen Theorembeweisern – Fallstudie –

342/03/99 A M. Bader, M. Schimper, Chr. Zenger: Hierarchical Bases for the Indefinite Helmholtz Equation

342/04/99 A Frank Strobl, Alexander Wisspeintner: Specification of an Elevator Control System

342/05/99 A Ralf Ebner, Thomas Erlebach, Andreas Ganz, Claudia Gold, Clemens Harlfinger, Roland Wism"uller: A Framework for Recording and Visualizing Event Traces in Parallel Systems with Load Balancing

342/06/99 A Michael Jaedicke, Bernhard Mitschang: The Multi-Operator Method: Integrating Algorithms for the Efficient and Parallel Evaluation of User-Defined Predicates into ORDBMS

342/07/99 A Max Breitling, Jan Philipps: Black Box Views of State Machines

SFB 342 : Methoden und Werkzeuge für die Nutzung paralleler Rechnerarchitekturen

Reihe B

342/1/90 B	Wolfgang Reisig: Petri Nets and Algebraic Specifications
342/2/90 B	Jörg Desel: On Abstraction of Nets
342/3/90 B	Jörg Desel: Reduction and Design of Well-behaved Free-choice Systems
342/4/90 B	Franz Abstreiter, Michael Friedrich, Hans-Jürgen Plewan: Das Werkzeug runtime zur Beobachtung verteilter und paralleler Programme
342/1/91 B	Barbara Paech1: Concurrency as a Modality
342/2/91 B	Birgit Kandler, Markus Pawlowski: SAM: Eine Sortier- Toolbox -Anwenderbeschreibung
342/3/91 B	Erwin Loibl, Hans Obermaier, Markus Pawlowski: 2. Workshop über Parallelisierung von Datenbanksystemen
342/4/91 B	Werner Pohlmann: A Limitation of Distributed Simulation Methods
342/5/91 B	Dominik Gomm, Ekkart Kindler: A Weakly Coherent Virtually Shared Memory Scheme: Formal Specification and Analysis
342/6/91 B	Dominik Gomm, Ekkart Kindler: Causality Based Specification and Correctness Proof of a Virtually Shared Memory Scheme
342/7/91 B	W. Reisig: Concurrent Temporal Logic
342/1/92 B	Malte Grosse, Christian B. Suttner: A Parallel Algorithm for Set-of-Support Christian B. Suttner: Parallel Computation of Multiple Sets-of-Support
342/2/92 B	Arndt Bode, Hartmut Wedekind: Parallelrechner: Theorie, Hardware, Software, Anwendungen
342/1/93 B	Max Fuchs: Funktionale Spezifikation einer Geschwindigkeitsregelung
342/2/93 B	Ekkart Kindler: Sicherheits- und Lebendigkeitseigenschaften: Ein Literaturüberblick
342/1/94 B	Andreas Listl; Thomas Schnekenburger; Michael Friedrich: Zum Entwurf eines Prototypen für MIDAS