A Functional Rephrasing of the Assumption/Commitment Specification Style*

Manfred Broy
Institut für Informatik
Technische Universität München
Postfach 20 24 20, 8 München 2, Germany

April 26, 1995

Abstract

The assumption/commitment (also called rely/guarantee) style has been advocated for the specification of interactive components of distributed systems. One of its motivations is to achieve modularity for state transition specifications of system components. It suggests the structuring of specifications into assumptions about the behavior of the component's environment and into commitments that are fulfilled by the component provided the environment fulfills these assumptions. We define the assumption/commitment formats for functional system specifications. In particular, we work out a canonical decomposition of system specifications following the assumption/commitment format into safety and liveness aspects. We demonstrate the format of assumption/commitment specifications by a number of examples. In particular, we discuss the methodological significance of the assumption/commitment format in the stepwise development of specifications.

*This work was partially supported by the Sonderforschungsbereich 342 "Werkzeuge und Methoden für die Nutzung paralleler Rechnerarchitekturen".

Contents

1 Introduction **3**

2 Concepts of Specification **5**

3 A First Simple Approach **7**
3.1 State-based Assumption/Commitment Specifications 7
3.2 A Simple Functional Assumption/Commitment Format 9
3.3 Analysis of Assumption/Commitment Specifications 12
3.4 A Canonical Assumption/Commitment Format 15

4 A More General Assumption / Commitment Format **16**
4.1 Assumption / Commitment Formats with Prophecies 16
4.2 Analysis of the Format with Prophecies 19

5 More Refined Semantic Concepts **21**
5.1 Additional Notational Concepts 21
5.2 Input Choice Specifications 22
5.3 Assumption and Commitment Specifications 24
5.4 Safety and Liveness 26
5.5 Component Safety and Liveness 27

6 Assumption/Commitment Specifications **30**

7 General Assumption/Commitment Format **31**

8 Conclusion **35**

1 Introduction

The precise and clear specification of the behavior of interactive system components is a decisive issue in systems engineering. For a composition-oriented *modular* specification of system components the assumption/commitment format (also called rely/guarantee format) has been advocated in numerous variations (see [Stølen 91], [Pandyá 90], among others). It provides a concept for the description of interfaces between interactive components and their environments. The basic idea of the assumption/commitment format consists in a clear separation of an interface specification of a component into the responsibilities of the component and those of its environment within their interaction.

Assumption/commitment formats of specifications are well-known and widely used for sequential noninteractive programs.[1] The semantics of noninteractive programs based on assignments expressing state changes can be modeled by relations between states or in the case of deterministic programs by functions mapping input (states) to output (states). For functions and similarly for relations on states specification techniques are used that restrict the input values or states of programs by predicates called *preconditions* and give specifications for the output values or output states by *postconditions* only for those input values (or input states) for which the preconditions are fulfilled. If the program is activated in a state that does not fulfill the precondition (the "assumption") then the specification does not restrict the output in any way.

For interactive programs this simple suggestive specification format using preconditions and postconditions is not immediately applicable. Interactive programs accept input step by step and successively produce output. In general, some of the output may be or even must be produced before the complete input history is available. This intermediate production of output generally depends on the intermediate provision of input and the intermediate nondeterministic decisions (taken so far). These considerations show that interactive systems are related to their environments in a more sophisticated way than noninteractive sequential systems are.

Often, distributed interactive systems are composed of a large number of interactive components. These components interact by updating shared store or by exchanging messages. By the interaction of the components the behavior of the system is generated. To achieve a modular system description we are interested not only in the description of the behavior of the entire system, but also in the description of the behavior of the individual components in isolation. It is the fundamental idea of modular system description that we can derive a description of the behavior of a composed system from the descriptions of the behaviors of its components.

The static and dynamic (logical) connection between a component of a distributed system and its environment is called its *interface*. An interface descrip-

[1] Preconditions as they are used in Hoare-logic may be seen as assumptions and postconditions as commitments.

tion is a, possibly incomplete, specification of those properties of a component that influence the overall behavior of a system that includes the component.

A suggestive and adequate way of describing interfaces of interactive components is of high interest in system development. The observations above show that interactive system components cannot simply be seen as functions or relations on states. More sophisticated representations are needed.

Operationally components of interactive systems can be seen as deterministic or nondeterministic input/output processing state machines. As well known, such operational views have draw-backs.

A nonoperational "functional" view of systems can be developed by modeling interactive system components by functions operating on streams of messages. Functional system specification techniques as outlined in [Broy 90] provide as *interface specifications* (sometimes also called *black box specifications*). In functional system specifications the system components' behaviors are specified in terms of the functional relationships between their input histories and their output histories. Nondeterministic system components are simply represented by sets of functions.

The assumption/commitment format for the specification of interactive systems (see for instance [Pandyá 90]), which is also called rely/guarantee paradigm (see for instance [Jones 83]), has been mainly studied in the framework of state-based system models. For these system models the assumption/commitment format has been developed for achieving compositionality for specification and verification techniques (see [Abadi, Lamport 90]). Functional system specification and verification techniques are compositional anyhow. Assumption/commitment formats therefore are not needed to achieve compositionality for functional specification techniques, but are of interest as a specification style with methodological advantages.

In the following we rephrase and discuss the assumption/commitment format for functional system specification techniques. We in particular show how we apply the assumption/commitment format in connection with an explicit decomposition of system specifications into safety and liveness properties. We give a number of examples and introduce a simple and a more involved format for the assumption/commitment specifications.

Based on the assumption/commitment format for functional system specifications we classify interactive systems as follows: components can be distinguished for which there exists

- a specification in a simple assumption/commitment format, where the assumptions about the input histories can be formulated independently of the output produced by the component,
- specifications in an only more sophisticated assumption/commitment format, where the assumptions on the input histories may depend partially on the output produced by the component so far.

In the first case assumption/commitment specifications are rather straightforward. In the second case a more involved format have to be used.

The paper is structured as follows. We give a short introduction into the basic structures of functional specification techniques. We repeat shortly the assumption/commitment format for state-oriented system specifications. Then we present a straightforward assumption/commitment format for functional system specifications. We analyze it and show why not all specifications of nondeterministic components can be given in this format. We then give a more sophisticated format and analyze it, too.

We are in particular interested in the classification of specifications and of specifying formulas and the usefulness of the assumption/commitment format in system specification and development.

2 Concepts of Specification

In this section we give a brief summary of the mathematical concepts of functional system models. We consider system components with a finite number of input and output channels. Over the channels messages are exchanged. A channel history is mathematically modeled by a stream of messages. The behavior of a (deterministic) component corresponds to a function mapping the streams associated with its input channels onto streams associated with its output channels.

A *stream* represents a communication history of a channel. A stream of messages over a given message set M is a finite or infinite sequence of messages. We define the set of streams M^ω by

$$M^\omega =_{def} M^* \cup M^\infty$$

By $x^\frown y$ we denote the result of concatenating two streams x and y. We assume that $x^\frown y = x$, if x is infinite. By $\langle\rangle$ we denote the empty stream.

If a stream x is a *prefix* of a stream y, we write $x \sqsubseteq y$. The relation $\sqsubseteq$ is called *prefix order*. It is formally specified as follows:

$$x \sqsubseteq y =_{def} \exists z \in M^\omega : x^\frown z = y$$

The behavior of deterministic interactive systems with n input channels and m output channels is modeled by functions

$$f : (M^\omega)^n \rightarrow (M^\omega)^m$$

called *(m, n)-ary stream processing functions*. We denote function application $f(x)$ often by $f.x$ to avoid brackets. A stream processing function is called *prefix monotonic*, if for all tuples of streams $x, y \in (M^\omega)^n$ we have

$$x \sqsubseteq y \Rightarrow f.x \sqsubseteq f.y$$

A stream processing function f is called *continuous*, if f is monotonic and for every directed set $S \subseteq M^\omega$ we have:

$$f.\sqcup S = \sqcup\{f.x : x \in S\}$$

By $\sqcup S$ we denote a least upper bound of a set S, if it exists. A set S is called *directed*, if for any pair of elements x and y in S there exists an upper bound in S. The set of streams is complete in the sense that for every directed set of streams there exists a least upper bound.

The behavior of deterministic interactive systems with n input lines and m output lines is modeled by stream processing functions

$$f : (M^\omega)^n \to (M^\omega)^m$$

where we assume that f is prefix continuous. The set of all stream processing functions of this form is denoted by

$$SPF_m^n$$

For simplicity we do not consider type information for the channels here and assume only that M is any set of messages.

By $SPEC_m^n$ we denote the set of all predicates

$$Q : SPF_m^n \to \mathbb{B}$$

The set $SPEC_m^n$ represents the set of all specifications for a component with n input channels and m output channels.

We often use the following functions on streams in specifications:

$rt : M^\omega \to M^\omega$	rest of a stream
$ft : M^\omega \to M \cup \{\bot\}$	first element of a stream
$\# : M^\omega \to \mathbb{N} \cup \{\infty\}$	length of a stream
$\copyright : \wp(M) \times M^\omega \to M^\omega$	filter of a stream

These functions are easily specified by the following axioms (let $x \in M^\omega, m \in M, S \in \wp(M)$):

$$\begin{array}{ll} rt.\langle\rangle = \langle\rangle, & rt(m^\frown x) = x, \\ ft.\langle\rangle = \bot, & ft(m^\frown x) = m, \\ \#\langle\rangle = 0, & \#(m^\frown x) = 1 + \#x, \end{array}$$

$$\begin{array}{ll} S\copyright\langle\rangle = \langle\rangle, & \\ S\copyright(m^\frown x) = m^\frown(S\copyright x), & \text{if } m \in S \\ S\copyright(m^\frown x) = S\copyright x, & \text{if } m \notin S \end{array}$$

These axioms specify the functions completely also on infinite streams under the assumption that the functions are prefix continuous. They are useful in proofs, too.

3 A First Simple Approach

In this section we introduce a first simple assumption/commitment format for specifications. We analyze questions of consistency of specifications in this format and study a canonical form of specifications in the assumption/commitment format. We start by rephrasing the assumption/commitment format for state based system models.

3.1 State-based Assumption/Commitment Specifications

For a state-based model of a system component a state space represented by a set $State$ of states is used. The system behavior is then given by the set of computations that the system can perform. A computation is represented by the sequences.

$$\sigma_0 \xrightarrow{E} \hat{\sigma}_0 \xrightarrow{C} \sigma_1 \ldots \sigma_i \xrightarrow{E} \hat{\sigma}_i \xrightarrow{C} \sigma_{i+1} \ldots$$

where $\sigma_i, \hat{\sigma}_i \in State$. Here the labels E and C on the arrows distinguish environment steps from component steps. We do not explicitly consider cases where several environment steps or component steps are carried out successively. Such multisteps can always be replaced by a single step.

Technically in a state-based system model a computation is a stream of triples of the form:

$$(\sigma_0, E, \hat{\sigma}_0)^\frown (\hat{\sigma}_0, C, \sigma_1)^\frown (\sigma_1, E, \hat{\sigma}_1)^\frown \cdots$$

By Com we denote the set of all computations.

Following the terminology of [Chandy, Misra 88] a predicate

$$Q : State \rightarrow \mathbb{B}$$

is called *stable* for a computation, if it holds for all successor states as long it holds for a given state. Mathematically expressed Q is stable for a computation if, for all $i \in \mathbb{N}$:

$$(Q(\sigma_i) \Rightarrow Q(\hat{\sigma}_i)) \wedge (Q(\hat{\sigma}_i) \Rightarrow Q(\sigma_{i+1}))$$

Of course, there are predicates that are stable with respect to the environment moves, but not with respect to the component moves or vice versa.

A stable predicate Q is called an *invariant* for a computation, if it holds for the initial state σ_0

$$Q(\sigma_0)$$

In a computation we have environment steps and component steps. Roughly speaking in an assumption/commitment specification for state based specifications certain steps of the environment and of the component are called correct and others are called incorrect. We use relations

$$R_S^E, R_S^C \subseteq State \times State$$

to represent the correct steps of a system and its environment. The relation R_S^E defines the correct steps of the environment. The relation R_S^C defines the set of correct steps of the component. Computations can nicely be understood as two person games played between the component and its environment. A computation is called a *win* (for the component), if either all its steps are correct or if the first incorrect step in the computation is an environment step.

Accordingly depending on the question by which computations we can reach a particular state we can distinguish following classes of states:

Good states: a good state is a state that is reachable from the initial state by correct moves, correct both for the environment and for the component,

Bad states: a bad state is a state that is reachable from the initial state by a sequence of arbitrary moves where the first incorrect move is done by the component,

Ugly states: an ugly state is a state that is reachable from the initial state by a sequence of arbitrary moves where the first incorrect move is done by the environment.

Of course, since, in general, each state may be reachable by different computations particular states may be good, bad, and ugly at the same time. This shows that assumption/commitment specifications need to be understood to characterize rather histories of computations than states.

In the assumption/commitment format of state-based system specifications restrictions in the form of commitments for the behavior of the component are formulated. These commitments are only required to be valid, however, as long as the assumptions (about the environment) are fulfilled. In an assumption/commitment format for state-based specifications we consider four predicates:

Environment safety (assumption): $R_S^E \subseteq State \times State$

Component safety (commitment): $R_S^C \subseteq State \times State$

Environment liveness (assumption): $P_L^E : Com \rightarrow \mathbb{B}$

Component liveness (commitment): $P_L^C : Com \rightarrow \mathbb{B}$

Of course P_L^E and P_L^C are assumed to be liveness conditions. The relations R_S^E and R_S^C are understood as stability requirements. From them we can schematically derive safety specifications for computations:

$$P_S^C, P_S^E : Com \rightarrow \mathbb{B}$$

by

$$P_S^E(t) = \forall \sigma, \hat{\sigma} : \{(\sigma, E, \hat{\sigma})\} ⓒ t \neq \langle\rangle \Rightarrow (\sigma, \hat{\sigma}) \in R_S^E$$
$$P_S^C(t) = \forall \hat{\sigma}, \sigma : \{(\hat{\sigma}, C, \sigma)\} ⓒ t \neq \langle\rangle \Rightarrow (\hat{\sigma}, \sigma) \in R_S^C$$

With these two predicates we may define an assumption/commitment format for the specification of state transition systems as follows. We define a safety property P_S and a liveness property P_L for the component by the following equations

$$P_S(t) = \forall t' : t' \sqsubseteq t \wedge P_S^E(t') \Rightarrow P_S^C(t')$$

and

$$P_L(t) = \left[P_S^E(t) \wedge P_L^E(t) \Rightarrow P_L^C(t)\right]$$

The assumption/commitment format of specifications for state transition systems exhibits a number of complications of technical and methodological nature:

1. a pure state-based view is not appropriate, since the proper continuation of behaviors (computations) of a system may depend on the question, how a particular state was reached (by legal or by illegal moves);

2. assumptions and commitments together form a system invariant; it is not obvious from a methodological point of view, whether one should first try to find this system invariant, and then later decompose it into invariants for the component and the environment or vice versa;

3. the separation of global liveness requirements of a system into liveness assumptions and commitments is not uniquely determined.

Clear answers have to be found to these three questions to justify the assumption/commitment format as a methodologically respectable concept.

Interestingly specifications in the assumption/commitment format for state transition systems are defined in terms of the system history (cf. [Abadi, Lamport 90]). Therefore it is suggestive to study them in a history based system model.

3.2 A Simple Functional Assumption/Commitment Format

A first simple assumption/commitment specification format for functional specifications is given in this section. The assumption/commitment format may be rephrased for interactive components of distributed systems informally as follows:

> "If the input history fulfills certain assumptions, then the component is committed to certain properties for its output history."

This informal phrase can be translated into predicate logic as follows. We work with two predicates

$$A : (M^\omega)^n \to \mathbb{B}$$
$$C : \big((M^\omega)^n \times (M^\omega)^m\big) \to \mathbb{B}$$

A is called the *assumption* and C is called the *commitment*. With the help of these two predicates we specify stream processing functions

$$f : (M^\omega)^n \to (M^\omega)^m$$

which model the behavior of a component by the following formula (for all input histories $x \in (M^\omega)^n$):

$$A.x \Rightarrow C(x, f.x)$$

Specifications of this form are said to be in the simple assumption/commitment format.

This format of specification is studied in detail in [Stølen et al. 92]. There, also a calculus is given that allows to reason about composed systems where all components are specified in the assumption/commitment format. Before we enter into a more formal analysis of specifications in the described assumption/commitment format we illustrate this format by a simple example.

Example: Bounded Buffer of Length one
A simple example for a specification in the assumption/commitment format is a bounded buffer of length one. A buffer is an interactive system that receives data messages and signals that indicate requests for output of data. Received data elements are stored until they are requested. A buffer of length one can store at most one data item. A buffer of length one works properly at least as long as data elements are never received when the buffer is full and request signals are never received when the buffer is empty. Only under these preconditions, under these *assumptions*, the buffer is *committed* to specific behavior. Then the output history of the component is properly reflecting the properties of a buffer.

Let D denote the set of data and $\textcircled{R}$ denote the request signal. We define the set of messages M as follows:

$$M = D \cup \{\textcircled{R}\}$$

In order to formalize the buffer's precondition the assumption predicate

$$A : M^\omega \to \mathbb{B}$$

has to express that data is not received when the buffer is full and request signals are not received when the buffer is empty. This is expressed formally by the predicate A specified as follows (for all $x \in M^\omega$):

$$A.x \equiv$$
$$\forall z \in M^\omega : z \sqsubseteq x \Rightarrow \#\big(\{\textcircled{R}\}\textcircled{C}z\big) \leq \#(D\textcircled{C}z) \leq \#\big(\{\textcircled{R}\}\textcircled{C}z\big) + 1$$

Here we denote by $\#x$ the number of elements in stream x and we use the filter function $\textcircled{c}$ as an auxiliary construct for the specification.

An equivalent and maybe more readable specification of A is given be the following three equations:

$$A.\langle\rangle \equiv true$$

$$A.\langle a\rangle \equiv (a \in D)$$

$$A(a^\frown b^\frown x) \equiv (a \in D \wedge b = \textcircled{R} \wedge A.x)$$

We extend A to infinite streams by the requirement

$$A.x = \forall z \in M^* : z \sqsubseteq x \Rightarrow A.z$$

The commitment predicate

$$C : (M^\omega \times D^\omega) \rightarrow I\!B$$

is specified by the following formula (for all $x \in M^\omega, y \in D^\omega$):

$$C(x,y) \equiv y \sqsubseteq D\textcircled{c}x \wedge \#y = \#\{\textcircled{R}\}\textcircled{c}x$$

This formula expresses that the output stream y is a prefix of the input stream x and the length of y is determined by the number of request signals in x. The commitment predicate alone is certainly not appropriate for specifying the behavior of the bounded buffer, since there does not exist a function f such that for all input streams x the commitment proposition

$$C(x, f.x)$$

holds. For instance, for the input stream $x = \langle\textcircled{R}\rangle$ (that does not fulfill the assumption) by the assertion

$$y \sqsubseteq D\textcircled{c}x$$

we deduce $y = \langle\rangle$ and by the assertion

$$\#y = \#\{\textcircled{R}\}\textcircled{c}x$$

we deduce $\#y = 1$ which gives a contradiction. This shows that the restriction of the commitment to those streams that fulfill the assumption is essential for ensuring the consistency of the specification. Using our format we obtain the following specification for the bounded buffer:

$$\mathit{Buffer}.f \equiv \forall x \in M^\omega : A.x \Rightarrow C(x, f.x)$$

The predicate *Buffer* characterizes the set of all functions that model behaviours of a component that we call a bounded buffer with capacity one. □

The example demonstrates that in the assumption/commitment format commitments are used essentially for the specification of system properties. These properties are weakened with the help of assumptions to avoid inconsistencies that may be caused by an overspecification in the commitments.

3.3 Analysis of Assumption/Commitment Specifications

In this section we analyze the consistency of assumption/commitment specifications of the simple format introduced above. An assumption/commitment specification of the form

$$A.x \Rightarrow C(x, f.x)$$

is called *consistent*, if there exists a continuous function f on streams that fulfills the specification.

If the assumption/commitment specification is consistent, then certainly for every legal input (every input that fulfills the assumption) there exists a legal output (an output that fulfills the commitment). In other words, the following formula holds:

$$\forall x : A.x \Rightarrow \exists y : C(x, y)$$

If this requirement is fulfilled, then we say that the assumption/commitment format with the assumption A and the commitment C is *compatible with output existence*. However, output existence compatibility is a necessary but not a sufficient condition for guaranteeing the consistency of the specification, since in addition to the assumption/commitment formula above we insist on f being a monotonic and continuous function.

The commitment part of an assumption/commitment specification is intended to constrain only the output history but not the input history of a system component. Taking into account the requirement of monotonicity we conclude that for every input history x for which

$$A.x$$

holds and for every output history[2] y for which

$$C(x, y)$$

holds and for all extensions $\tilde{x}$ of the input history x (which are streams $\tilde{x}$ with $x \sqsubseteq \tilde{x}$) there exists an extension $\tilde{y}$ of the output stream y (mathematically expressed $y \sqsubseteq \tilde{y}$) such that the following proposition holds:

$$A.\tilde{x} \Rightarrow C(\tilde{x}, \tilde{y})$$

If the assumption A and the commitment C fulfill this requirement, then we say that the assumption/commitment format with assumption A and commitment C is *compatible with monotonicity*. This condition asserts that for every input history that fulfills the assumption and every output history that fulfills the commitment and every extension of the input history to some input history that fulfills the assumption there exists an extension of the output history that fulfills the assumption.

[2]Such an output history always exists, if the specification is compatible with output existence.

As a final aspect of the consistency of specifications in the assumption/commitment format we consider continuity. For achieving consistency, it is required that assumption/commitment specifications are compatible with continuity requirements for the specified functions f in the following sense. A simple assumption/commitment specification is compatible with continuity requirements, if for every pair of chains $\{x_i : i \in I\!N\}$ and $\{y_i : i \in I\!N\}$ whenever for all $i \in I\!N$

$$A.x_i \Rightarrow C(x_i, y_i)$$

we have

$$A.x \Rightarrow C(x, y)$$

for $x = \sqcup\{x_i : i \in I\!N\}$ and $y = \sqcup\{y_i : i \in I\!N\}$. If the assumption A and the commitment C fulfill this requirement, then we say that the assumption/commitment format with assumption A and commitment C is *compatible with continuity.*

Theorem 1 *If a specification in the assumption/commitment format is compatible with output existence, monotonicity and continuity, then it is consistent.*

Proof: Let us assume that an assumption A and a commitment C are given such that in the assumption/commitment format is compatible with output existence, monotonicity and continuity. We construct a chain of functions f_i whose the least upper bound fulfills the corresponding assumption/commitment specification. We define the functions f_i inductively such that for all $i \in I\!N$:

$$A.x \wedge \#x < i \Rightarrow C(x, f_i.x)$$

We define the function f_0 trivially by (for all streams x):

$$f_0.x = \langle\rangle$$

Given f_i we define f_{i+1} as follows:

$$\#x < i \Rightarrow f_{i+1}.x = f_i.x$$

$$\#x = i \wedge \neg A.x \Rightarrow f_{i+1}.x = f_i.x$$

if $\#x = i$ and $A.x$ holds then, since the assumption and the commitment are compatible with monotonicity (in the case $i = 0$ according to output compatibility), there exists some output y such that $C(x, y)$. We define:

$$f_{i+1}.x = y$$

Finally we define:

$$\#x > i \wedge z \sqsubseteq x \wedge \#z = i \Rightarrow f_{i+1}.x = f_{i+1}.z$$

By continuity compatibility it follows that the least upper bound $\sqcup\{f_i : i \in I\!N\}$ fulfills the specification in the assumption/commitment format. This can be shown as follows: given an input history x we define a chain $\{x_i : i \in I\!N\}$ by the assertion

$$x_i \sqsubseteq x \wedge \#x_i = min(\#x, i)$$

By the construction of the f_i we obtain:

$$A(x_i) \Rightarrow C(x_i, f_{i+1}.x_i)$$

Therefore by continuity compatibility we have:

$$A(\sqcup\{x_i : i \in I\!N\}) \Rightarrow C(\sqcup\{x_i : i \in I\!N\}, \sqcup\{f_{i+1}.x_i : i \in I\!N\})$$

and hence we obtain:

$$A(x) \Rightarrow C(x, f.x)$$

with $x = \sqcup\{x_i : i \in I\!N\}$ and $f = \sqcup\{f_i : i \in I\!N\}$. This proves the consistency of the specification. □

There is a remarkable consequence of the requirement of monotonicity for the functions specified in the assumption/commitment format. If we have $\neg A.x$ for some input history x, which means that the input history x does not fulfill the assumptions, it seems that nothing can be predicted for the output $f.x$ according to the assumption/commitment format. If in an implicative formula the premise is false, then the formula is true. However, implicitly something may be concluded about the output $f.x$ even in some cases where x does not fulfill the assumption. Let f be a stream processing function that fulfills the formula

$$A.x \Rightarrow C(x, f.x) \qquad (*)$$

Then even, if we have $\neg A.x$ for some input history x, as long as we have $A.\tilde{x}$ for some input history $\tilde{x}$ with $x \sqsubseteq \tilde{x}$, we can expect $C(\tilde{x}, f.\tilde{x})$ and by the monotonicity we know $f.x \sqsubseteq f.\tilde{x}$. This shows that the monotonicity of f induces some implicit requirements by (*) onto the output history $f.x$ even for those input histories x that do not fulfill the assumption predicate in cases where x is a prefix of an input history $\tilde{x}$ that fulfills the assumption A. This reflects exactly what we expect for an interactive component: for every prefix of an input history $\tilde{x}$ for which the input assumption is valid the output fulfills the commitment. This output is a prefix of the output to the extended input.

These implicit requirements for input histories that do not fulfill the assumptions arises only in assumption/commitment specifications with assumptions A that do hold for certain input histories $\tilde{x}$ but do not hold for some of their prefixes x. We call an assumption A *explicit*, if for all input histories $\tilde{x}$ the following assertion holds:

$$A.\tilde{x} \wedge x \sqsubseteq \tilde{x} \Rightarrow A.x$$

Certainly assumption/commitment specifications that use assumptions that are safety predicates (in the sense of [Dederichs, Weber 90]) are always explicit. However, there are explicit assumptions that are not safety predicates. A safety predicate holds for a stream, if it holds for all its finite prefixes (a more formal definition of the concept of a safety predicate is given in section 5.4). In contrast, an assumption in an explicit assumption/commitment specification may not hold for an infinite stream, although it holds for all its finite approximations.

In the case of explicit specifications in the assumption/commitment format the proof of the theorem above can be slightly simplified.

3.4 A Canonical Assumption/Commitment Format

In this section we construct a canonical form of the simple assumption/commitment format for a given functional system specification. For doing that, we study the decomposition of a specifying predicate P into assumptions and commitments. The proposition

$$P.f$$

characterizes the behavior of a system component in terms of a set of functions f. We want to decompose the predicate P into predicates A and C such that $P.f$ can be replaced by the assumption/commitment format

$$P.f \equiv \forall x : A.x \Rightarrow C(x, f.x) \qquad (**)$$

The predicates in this format are not uniquely determined and, in general, do not even exist. An example, where such a format does not exist for a specification and therefore this simple decomposition does not work, is given by the unreliable one-element buffer in section 4.1.

However, we may think about a canonical decomposition of a function specification into assumptions and commitments in cases where such a format exists. We define predicates

$$A.x \equiv \neg\forall y : \exists f : P.f \wedge y = f.x$$

$$C(x, y) \equiv \exists f : P.f \wedge y = f.x$$

If with these definitions the formula (**) is valid, then we say that P *can be decomposed into a simple assumption/commitment scheme.*

Deterministic components[3] can always be specified by the introduced assumption/commitment format. This has been discussed in detail in [Stølen et al. 92]. For nondeterministic components such a decomposition is not always possible. There are situations where the simple assumption/commitment format as introduced above does not work. Examples are nondeterministic system

[3] More precisely components which show a deterministic behavior on input that fulfills the assumption.

components where the input assumption depends on the output produced so far for certain prefixes of input histories. For instance, consider an unreliable one-element buffer that sometimes loses its incoming data and therefore indicates by signals whether it has received its input data properly or not. Here the assumption that input is never received when the buffer is full does depend not only on the input history but also on the actual output history and therefore cannot be written as a simple predicate on the input stream x. For dealing with such situations we develop a more general assumption/commitment format in the following section.

4 A More General Assumption / Commitment Format

As pointed out the simple assumption/commitment format introduced in the previous section does not work for nondeterministic components where the input assumption also depends on the output history produced so far. For those components we have to take into account the output produced for the corresponding input history to restrict the further input.

4.1 Assumption / Commitment Formats with Prophecies

We start with a simple example that does not fit into the assumption/commitment format introduced in the previous section to illustrate the complications.

Example: One-element lossy buffer
A one-element lossy buffer is a component that may store at most one data element. It receives input messages which are either data elements or requests (represented by the signal ®). If the buffer never gets a request signal when it is empty and never gets a data message when it is full then it behaves properly like a one-element buffer, or it may be lossy. It may lose a data message that is sent to it in an empty state, but such a loss is indicated by the signal ®; if it stores its data message correctly this is indicated by the signal $\surd$.

We use the following sets of messages

$$M = D \cup \{®\}$$

$$N = D \cup \{®\} \cup \{\surd\}$$

We specify functions

$$f : M^\omega \to N^\omega$$

with the help of the predicate:

$$P : \big((D \cup \{\emptyset\}) \to (M^\omega \to N^\omega)\big) \to \mathbb{B}$$

where the set $D \cup \{\emptyset\}$ is used to denote the state of the buffer. Initially the buffer is in the state "empty". The behavior of the lossy buffer is characterized by a predicate

$$Q : (M^\omega \to N^\omega) \to I\!B$$

specified by the formula:

$$Q.f \equiv \exists h : f = h.\emptyset \wedge P.h$$

The auxiliary predicate P is specified as follows. If the component is empty and it receives a data message, it either stores the data message and acknowledges this by sending the signal $\surd$ or it loses it and indicates this by the signal Ⓡ. We define P by the following equations:

$$\begin{aligned} P.h \equiv \quad \forall x \in M^*, d \in D : \quad & \big((h.\emptyset).(\langle d\rangle^\frown x) = \langle\surd\rangle^\frown(h.d).x \vee \\ & (h.\emptyset).(\langle d\rangle^\frown x) = \langle Ⓡ\rangle^\frown(h.d).x\big) \\ & \wedge \\ & \big((h.d).(\langle Ⓡ\rangle^\frown x) = \langle d\rangle^\frown(h.\emptyset).x \vee \\ & (h.d).(\langle Ⓡ\rangle^\frown x) = \langle Ⓡ\rangle^\frown(h.\emptyset).x\big) \end{aligned}$$

Obviously we cannot use the simple assumption/commitment format as introduced above to specify the component described by Q. If we just consider an input history we cannot observe whether the buffer is empty or full after it has received that messages. This can only be recognized when looking in addition to the input also to the output history. These considerations show that for the unreliable buffer there does not exist a simple assumption predicate that is independent of the output history. □

The example demonstrates that for particular component specifications we have to refer to the output that has been produced so far by the component in the assumption predicate, too. This leads to the following generalized assumption/commitment format:

$$A(x, f.x) \Rightarrow C(x, f.x)$$

However, in this format there is no longer any syntactic difference between assumptions and commitments, since both equally depend on the input and the output history. Nevertheless we can write assumption/commitment specifications in that format.

Example: One-element lossy buffer (continued)
We define the assumption

$$A : M^\omega \times N^\omega \to I\!B$$

as follows: let A be the weakest predicate that fulfills the following equations (for all $x \in M^\omega, y \in N^\omega$):

$$\begin{array}{ll}
A(\langle\rangle, y) & \equiv true \\
A(x, \langle\rangle) & \equiv (ft.x \in D) \\
A(\langle \circledR \rangle^\frown x, y) & \equiv false \\
A(\langle d \rangle^\frown x, \langle \circledR \rangle^\frown y) & \equiv A(x, y) \\
A(\langle d \rangle^\frown \langle d' \rangle^\frown x, \langle \surd \rangle^\frown y) & \equiv false \\
A(\langle d \rangle^\frown \langle \circledR \rangle^\frown x, \langle \surd \rangle^\frown y) & \equiv A(x, rt.y)
\end{array}$$

The commitment predicate

$$C : M^\omega \times N^\omega \to \mathbb{B}$$

is specified by the weakest predicate that fulfills the following equations:

$$\begin{array}{lcl}
C(\langle\rangle, y) & \equiv & (y = \langle\rangle) \\
C(\langle d \rangle^\frown x, y) & \equiv & (ft.y = \circledR \quad \wedge \quad C(x, rt.y)) \\
 & & \quad \vee \\
 & & (ft.y = \surd \quad \wedge \\
 & & (ft.x = \circledR \quad \Rightarrow \quad (ft.rt.y = d \quad \wedge \quad C(rt.x, rt.rt.y))))
\end{array}$$

We put together the assumption A and the commitment C to obtain a specification Q of the lossy buffer in the assumption/commitment format:

$$Q.f \equiv \forall x : A(x, f.x) \Rightarrow C(x, f.x)$$

The assumption A and the commitment C have the characteristic properties that the assumption does only constrain the input history and the commitment does only constrain the output history. □

It is the fundamental idea of the assumption/commitment format that the assumption determines which input history fulfills the assumption, while the commitment characterizes for the input histories that fulfills the assumption which properties the output histories have to fulfill.

One logical trick to make sure that assumptions do not constrain the output history and commitments do not constrain the input histories are prophecies[4]. A prophecy can be understood as a hypothesis about the output histories. The following more general assumption/commitment format uses a prophecy variable $\hat{f}$:

$$Q.f \equiv \exists \hat{f} : \forall x : A(x, \hat{f}.x) \Rightarrow f = \hat{f} \wedge C(x, f.x)$$

Intuitively the format can be explained as follows: in the assumption we use a particular hypothesis about the behavior of the system component represented by the prophecy $\hat{f}$. We formulate the assumption based on this hypothesis. If there is at least one hypothesis for which the assumption is fulfilled, then the component fulfills its commitment. A more careful analysis of the assumption/commitment format with prophecies is given in the following section.

[4] In our case it may be more appropriate to speak of a hypothesis instead of a prophecy.

4.2 Analysis of the Format with Prophecies

For analyzing the assumption/commitment format that works with prophecies we consider two cases. Let us first assume that there exists a prophecy $\hat{f}$ for which the assumption is always wrong. In mathematical terms we assume:

$$\exists \hat{f} : \forall x : \neg A(x, \hat{f}.x)$$

Then obviously the formula

$$\exists \hat{f} : \forall x : A(x, \hat{f}.x) \Rightarrow f = \hat{f} \wedge C(x, f.x)$$

is trivially true, since we can choose the prophecy $\hat{f}$ such that the premise is always false.

Now we assume for all prophecies that there exist input histories that fulfill the assumption. Mathematically expressed we assume

$$\forall \hat{f} : \exists x : A(x, \hat{f}.x)$$

Since the subformula $f = \hat{f}$ does not depend on the identifier x, we can move it under the given assumption out of the scope of the universal quantifier and we replace the formula

$$\exists \hat{f} : \forall x : A(x, \hat{f}.x) \Rightarrow f = \hat{f} \wedge C(x, f.x)$$

by the logically identical formula:

$$\exists \hat{f} : f = \hat{f} \wedge \forall x : A(x, \hat{f}.x) \Rightarrow C(x, f.x)$$

This formula, however, is by the rules of equational logic equivalent to the formula:

$$\forall x : A(x, f.x) \Rightarrow C(x, f.x)$$

Our analysis leads to a simple assumption/commitment scheme where the assumption fulfills the following notion of properness. We call a predicate:

$$A : (M^\omega \times N^\omega) \rightarrow \mathbb{B}$$

a *proper* assumption, if

$$\forall \hat{f} : \exists x : A(x, \hat{f}.x)$$

Given a proper assumption A and a commitment predicate:

$$C : (M^\omega \times N^\omega) \rightarrow \mathbb{B}$$

a specification:

$$Q.f \equiv \forall x : A(x, f.x) \Rightarrow C(x, f.x)$$

is said to be in the general assumption/commitment format. This simplification of the format with prophecies is possible, since we assume that the proposition

$$\forall \hat{f} : \exists x : A(x, \hat{f}.x)$$

holds. If this proposition would not hold, the specification in the assumption/commitment format with a prophecy which is of the form

$$Q.f \equiv \forall \hat{f} : A(x, \hat{f}.x) \Rightarrow f = \hat{f} \wedge Q(x, f.x)$$

is trivially identical to $Q.f \equiv true$.

The assumption/commitment format with explicit and proper assumptions is certainly not uniquely determined for the specification of a given component. Any assumption A that we can decompose by

$$A(x, y) \equiv A_1(x, y) \ \wedge \ A_2(x, y)$$

allows also to write

$$A_1(x, f.x) \Rightarrow \big(A_2(x, f.x) \Rightarrow C(x, f.x)\big)$$

instead of

$$A(x, f.x) \Rightarrow C(x, f.x)$$

and thus leads to a specification in the assumption/commitment format with a possibly weaker assumption.

Based on this observation we may look for the strongest assumption. Given a specification

$$Q.f \equiv \forall x : S(x, f.x)$$

we define such an assumption A by the following proposition

$$A(x, y) \equiv \neg \forall f, z : S(x \frown z, y \frown f.z)$$

This gives us the strongest assumption for Q. We define a commitment

$$C(x, y) \equiv A(x, y) \ \wedge \ S(x, y)$$

The above format for assumption/commitment specifications with proper assumptions works for the specification of most of but not all components. For instance, the well-known Brock-Ackermann anomaly (cf. [Brock, Ackermann 81]) still can arise if we restrict ourselves to specifications of the assumption/commitment format above. The reason is obvious: the format above basically is a relational specification.

This problem is overcome in the approach proposed in [Stølen et al. 92]. However, we want to go even beyond that approach in the following and deal with components in which the complications of nonstrict fair merge arise. Therefore we generalize the format once more. We follow the concept of input choice specifications as introduced in [Broy 90]. This concept will be used in the assumption/commitment format.

5 More Refined Semantic Concepts

In this section we deal with three structuring concepts along the following lines:

- structuring of system specifications into specifications about a particular component and its environment,
- structuring of system specifications into safety and liveness properties,
- treating more complex forms of nondeterminism.

In the beginning we introduce a number of notations and some more sophisticated semantic concepts including classifications of specifications into safety and liveness properties.

Also in the following we structure system descriptions into requirements about the environment and those about the component leads to a more elaborate assumption/commitment format.

In connection with safety and liveness characterizations we may also decompose assumptions and commitments into safety and liveness properties.

There are components the nondeterministic behavior of which cannot be modelled by a set of continuous functions where for a given input history one of these functions is chosen and applied to the given input history. Sometimes to guarantee more sophisticated liveness propertics the choice of one of the functions may depend on the input history. The behavior of such components can be described by input choice specifications which represent relations between stream processing functions and input histories.

5.1 Additional Notational Concepts

In the following we introduce some further mathematical and notational concepts and abbreviations that will make it simpler to talk about more refined concepts of assumption/commitment specifications.

Let us introduce following abbreviations: for an arbitrary element $x \in S$ where $(S, \sqsubseteq)$ is a partial order we define the set $\downarrow x \subseteq S$ called the *downward closure* of x by the equation

$$\downarrow x = \{z \in S : z \sqsubseteq x\}$$

A set $\{x_i \in S : i \in I\!N\}$ is called a *chain*, if for all $i \in I\!N$ we have $x_i \sqsubseteq x_{i+1}$.

Given a stream processing function

$$f : (M^\omega)^n \rightarrow (M^\omega)^m$$

we define for an input history $z \in (M^\omega)^n$ the function

$$f \Downarrow z : (M^\omega)^n \rightarrow (M^\omega)^m$$

by

$$(f \Downarrow z).x \equiv \sqcup\{f.\tilde{x} : \tilde{x} \sqsubseteq x \wedge \tilde{x} \sqsubseteq z\}$$

The function $f \Downarrow z$ represents the "continuous restriction" of the function f to the elements in the set $\downarrow z$. We define the function

$$f\check{\Downarrow}z : (M^\omega)^n \rightarrow (M^\omega)^m$$

by

$$(f\check{\Downarrow}z).x \equiv \sqcup\{f.\tilde{x} : \tilde{x} \sqsubseteq x \wedge \tilde{x} \sqsubset z\}$$

We write $x \sqsubset z$ for $x \sqsubseteq z \wedge x \neq z$. The function $f\check{\Downarrow}z$ denotes the continuous restriction of the function f to the elements in $\{x \in S : x \sqsubset z\}$.

A tuple of streams is called *finite*, if all its streams are finite. A function on streams is called *output finite*, if its output is always finite. For an arbitrary set S of streams or functions on streams we denote by $FIN[S]$ its subset of finite elements or output finite functions.

Given a specification $Q \in SPEC^n_m$, a pair of chains

$$(\{x_i \in (M^\omega)^n : i \in I\!N\}, \{y_i \in (M^\omega)^m : i \in I\!N\})$$

are called an *observation* about (the component spezified by) Q, if there exists a function f with $Q.f$ such that for all $i \in I\!N$:

$$y_i \sqsubseteq f.x_i$$

and in addition

$$\sqcup\{y_i : i \in I\!N\} = \sqcup\{f.x_i : i \in I\!N\}$$

The behavior of a system component specified by a predicate Q can also be represented by the set of all observations about Q.

5.2 Input Choice Specifications

The behavior of a component can be described by a set of observations. Every specification

$$Q : \big((M^\omega)^n \rightarrow (M^\omega)^m\big) \rightarrow I\!B$$

defines a set $Obs(Q)$ of observations by the following definition:

$$\big(\{x_i : i \in I\!N\}, \{y_i : i \in I\!N\}\big) \in Obs(Q)$$

$$\Leftrightarrow$$

$$\exists f : Q.f \ \wedge \ \forall i \in I\!N : y_i \sqsubseteq f.x_i \ \wedge \ f\big(\sqcup\{x_i : i \in I\!N\}\big) = \sqcup\{y_i : i \in I\!N\}$$

Unfortunately, there are cases where the set of observations for a component cannot be described by a set of functions characterized by a predicate Q. We illustrate this statement by an example.

Example: Arbiter
An arbiter is a component that accepts as input data messages taken from a set D and produces as output messages taken from $D \cup \{|\}$.

We specify the behavior of the arbiter by defining its set of observations. A pair of chains

$$\big(\{x_i \in D^\omega : i \in I\!N\}, \{y_i \in (D \cup \{|\})^\omega : i \in I\!N\}\big)$$

is an observation about the arbiter, if and only if, with $y = \sqcup\{y_i : i \in I\!N\}$, the following assertions hold:

$$D \copyright y_i \sqsubseteq x_i$$
$$\{|\} \copyright y = |^\infty$$
$$D \copyright y = \sqcup\{x_i : i \in I\!N\}$$

These assertions specify that the arbiter eventually reproduces all elements in its input stream and also inserts an infinite number of copies of the signal $|$.

The behavior of the arbiter cannot be simply described by a set of continuous functions that fulfill a predicate

$$Q : \big(D^\omega \to (D \cup \{|\})^\omega\big) \to I\!B$$

since due to the observations above for a function f which characterizes the behavior of the arbiter we require for instance

$$f.\langle\rangle = |^\infty$$

and

$$f.\langle d\rangle = |^k \frown d \frown |^\infty$$

for some $k \in I\!N$ which contradicts the monotonicity requirement for f. By a set of monotonic functions we cannot express the liveness property that the number of signals $\{|\}$ in the output stream is required to be infinite without getting into a conflict with the monotonicity requirement. □

Components of the type of the arbiter can be specified nevertheless using a set of functions by so-called input choice or input constraint specifications. A predicate

$$P : ((M^\omega)^n \to (M^\omega)^m) \times (M^\omega)^n \to I\!B$$

is called an *input choice or input constraint specification*. The set of input choice specifications is abbreviated by ICS^n_m. An input choice specification P characterizes a function f with respect to a given input x by $P(f, x)$ (for a more detailed methodological justification of the concept of input choice specifications, see [Broy 90]).

An input choice specification P characterizes the behavior of a component as follows. A pair of chains (X, Y) is called an observation about the component specified by P, if there exists a function f such that $P(f, \sqcup X)$ and for all $i \in I\!N$:

$$y_i \sqsubseteq f(x_i)$$

and

$$\sqcup Y = f(\sqcup X)$$

To illustrate the concept we give an input choice specification for the arbiter component.

Example: Arbiter
An arbiter is a component that receives messages from a set D and produces messages from the set $D \cup \{|\}$ where we assume that $| \notin D$.

The partial correctness of the arbiter, its safety property, is desribed by the predicate

$$Q : \left[D^\omega \rightarrow (D \cup \{|\})^\omega\right] \rightarrow I\!B$$

specified by

$$Q.f \equiv \forall x \in D^\omega : D \textcircled{c} f.x \sqsubseteq x$$

The input choice specification of the arbiter is given by the following equation:

$$P(f, x) \equiv Q.f \wedge D \textcircled{c} f.x = x \wedge \#\{|\} \textcircled{c} f.x = \infty$$

The predicate Q formalizes the safety properties of the component. To obtain P from Q the liveness properties with respect to input x are added. □

Input choice specification allow to specify the behavior of certain nondeterministic components that cannot be specified simply by predicates on functions. We use input choice specifications to describe a general assumption/commitment format.

It does not make much sense to write a specification for the arbiter in the assumption/commitment format, since the arbiter is supposed to restrict the output for all input histories. In other words, the assumption is always fulfilled.

5.3 Assumption and Commitment Specifications

In this section we define a general assumption/commitment format for input choice specifications. We characterize which properties of input choice specifications are appropriate for component specifications.

An input choice specification $P \in ICS_m^n$ is called *proper*, if the following three conditions are fulfilled:

- *basic consistency (output existence)*: for every input history there exists at least one specified behavior:

$$\forall x : \exists f : P(f, x)$$

- *consistent continuation of finite observations (monotonicity)*: for every pair (f, x) where $P(f, x)$ holds and every output finite approximation $\bar{f} \in FIN[SPEC_m^n]$ of f where $\bar{f} \sqsubseteq f$ and every input history $\tilde{x}$ with $x \sqsubseteq \tilde{x}$ there exists a continuous function $\tilde{f}$ with $\bar{f} \sqsubseteq \tilde{f}$ such that $P(\tilde{f}, \tilde{x})$. More formally, this reads:

$$P(f,x) \wedge x \sqsubseteq \tilde{x} \wedge \bar{f} \in FIN[SPEC_m^n] \wedge \bar{f} \sqsubseteq f \Rightarrow \exists \tilde{f} : \bar{f} \sqsubseteq \tilde{f} \wedge P(\tilde{f}, \tilde{x})$$

- *continuity consistency*: for chains $\{x_i \in (M^\omega)^n : i \in I\!N\}$ and functions $\{f_i \in (M^\omega)^n \rightarrow (M^\omega)^m : i \in I\!N\}$ with $P(f_i, x_i)$ for all $i \in I\!N$ we have

$$P\big(\sqcup \{f_i \in (M^\omega)^n \rightarrow (M^\omega)^m : i \in I\!N\}, \sqcup \{x_i \in (M^\omega)^n : i \in I\!N\}\big)$$

Only input choice specifications that are proper and therefore have the three reqired properties are considered in the following as acceptable specifications.

Every input choice specification that does not fulfill the first condition can be turned into one that fulfills the first condition by means of the *chaotic closure*. The chaotic closure $CC(P)$ of an input choice specification P is defined as follows:

$$CC(P)(f,x) \equiv \big[(\exists \tilde{f} : P(\tilde{f}, x)) \Rightarrow P(f,x)\big]$$

The formula defining the chaotic closure already shows a format of an assumption/commitment specification. In the following we characterize which kind of input choice specifications can serve as pure assumptions or which kind can serve as pure commitments.

A specification $P \in ICS_m^n$ is called a pure *assumption* (about the environment), if it includes constraints for the input history that may depend only on output produced for streams $\tilde{x}$ with $\tilde{x} \sqsubset x$. Formally expressed, if the following formula holds:

$$P(f,x) = P(f \Downarrow x, x)$$

The formula expresses that the proposition $P(f, x)$ does not include any restrictions on the output of f for input x but at most depends on the output of f for input streams $\tilde{x}$ where $\tilde{x} \sqsubset x$. As the formula indicates the validity of $P(f, x)$ for an assumption does not depend on the output history produced by f for the input history x in addition to the output history produced for some input history $z \sqsubseteq x$ for which $P(f, z)$ holds, but just on the choice of the continuation $\tilde{z}$ of the input history z where $z^\frown \tilde{z} = x$.

An input choice specification P is called a *pure commitment*, if there does not exist a nontrivial pure assumption Q such that

$$P(f,x) \equiv [Q(f,x) \Rightarrow P(f,x)]$$

In analogy to the definition for the simple assumption/commitment format an assumption $P \in ICS_m^n$ is called *explicit*, if the following formula holds:

$$P(f,x) \Rightarrow [\forall z : z \sqsubseteq x \Rightarrow P(f,z)]$$

This formula expresses that if the history x is a "correct" input for f with respect to the assumption then all its prefixes are correct for f, too.

Explicit environment specifications are of interest in connection with the goal to split specifications into safety and liveness properties.

5.4 Safety and Liveness

In this section we extend the notion of safety conditions and liveness conditions to input choice specifications. As it has been pointed out several times before, safety conditions are those conditions that restrict and characterize the finite observations about a system. In contrast to safety conditions, liveness conditions guarantee that certain observations (about the output) eventually can be made. Hence they restrict the behavior of a system in addition to the safety properties.

An input choice specification $P \in ICS_m^n$ is called a (pure) *safety condition*, if

$$P(f,x) \equiv \forall \tilde{x} \in FIN[\downarrow x], \tilde{f} \in FIN[\downarrow f] : P(\tilde{f}, \tilde{x})$$

This definition just reflects the well-known concept that safety conditions are those properties that are fulfilled for given elements, if and only if they are fulfilled for all their finite approximations.

An input choice specification $P \in ICS_m^n$ is called (pure) *liveness condition*, if

$$\forall x \in FIN[(M^\omega)^n], f \in FIN[SPF_m^n] :$$
$$\exists \tilde{x} \in (M^\omega)^n, \exists \tilde{f} \in SPF_m^n : x \sqsubseteq \tilde{x} \wedge f \sqsubseteq \tilde{f} \wedge P(\tilde{f}, \tilde{x})$$

This definition of liveness just follows the well-known concept that a property is a liveness condition, if for given finite elements there are always elements, for which the condition is fulfilled and that are approximated by the given finite elements. In other words every finite computation history can be resumed into a computation history that fulfills the liveness conditions.

As well known, every input choice specification $P \in ICS_m^n$ can be decomposed canonically into a safety condition P_S and a liveness condition P_L such that

$$P(f,x) = P_L(f,x) \wedge P_S(f,x)$$

The safety predicate P_S is defined schematically along the lines of the definition of safety given above as follows:

$$P_S(f,x) \equiv \forall \bar{x} \in FIN[\downarrow x], \bar{f} \in FIN[\downarrow f] : \exists \tilde{f}, \tilde{x} : P(\tilde{f}, \tilde{x}) \wedge \bar{x} \sqsubseteq \tilde{x} \wedge \bar{f} \sqsubseteq \tilde{f}$$

This is a straightforward definition reflecting the fact that an element fulfills the safety condition contained in an arbitrary predicate P, if every finite approximation of the element fulfills the safety condition contained in P. A finite element $\bar{y}$ fulfills the safety condition contained in P, if there exists an element $\tilde{y}$ such that $\bar{y} \sqsubseteq \tilde{y}$ and $P.\tilde{y}$.

The liveness condition P_L contained in the predicate P can be easily specified with the help of the safety condition P_S by the following formula:

$$P_L(f,x) \equiv (P_S(f,x) \Rightarrow P(f,x))$$

Given an input choice specification $P \in ICS_m^n$ and its split into its safety condition P_S and its liveness condition P_L we can distinguish the following four situations for a pair (f,x) consisting of a behavior function f and an input history x:

$P(f,x)$	$P_S(f,x)$	$P_L(f,x)$	Case
tt	tt	tt	1.
ff	tt	ff	2.
ff	ff	tt	3.
ff	ff	ff	4.

The intuitive explanations for these cases read as follows:

1. For the input history x the output behavior produced by the function f is safe and live.

2. For the input history x the output behavior produced by the function f is safe, but not live, it may become live by adding further output.

3. For the input history x the output behavior produced by the function f is not safe and therefore trivially live.

4. Does not occur.

From a methodological point of view, it may not make much sense to consider arbitrary liveness conditions independent from safety conditions (see [Dederichs, Weber 90]). Therefore we are, in practice, interested in the validity of the liveness predicate P_L only for those pairs (f,x) for which the safety condition $P_S(f,x)$ holds.

5.5 Component Safety and Liveness

Given an input choice specification for an input history x and a behavior function f we may separate safety and liveness conditions both for the component and its environment.

An input choice specification $P \in ICS_m^n$ is called a *commitment safety condition*, if the following formula is valid:

$$P(f,x) \equiv \forall \tilde{f} \in FIN[\downarrow f] : P(\tilde{f},x)$$

In other words a specification is a commitment safety condition, if it is true for a function f if and only if it is true for all its output finite approximations.

Example: Commitment safety condition
The condition (let $f \in SPF_1^1$ and $x \in M^\omega$)

$$\#f.x \leq \#x$$

is a commitment safety condition. It expresses that the component never produces a larger number of output messages than its number of received input messages. □

An assumption specification P is called an *assumption safety condition*, if the following formula is valid:

$$P(f,x) \equiv \forall \tilde{x} \in FIN[\downarrow x] : P(f,\tilde{x})$$

In other words, an assumption specification is an assumption safety condition, if it is fulfilled for an input history x if and only if it is fulfilled for all finite approximations of x.

Example: Assumption safety condition
A typical assumption safety condition is given by the following formula (let $f \in SPF_1^1$ and $x \in M^\omega$):

$$\forall \tilde{x} : \tilde{x} \sqsubset x \Rightarrow \#\tilde{x} \leq \#f.\tilde{x}$$

This specification expresses the following assumption about the environment: the environment never sends more input messages than output messages are produced so far. □

A component specification P is called a *commitment liveness condition*, if

$$\forall f \in FIN[SPF_m^n] : \exists \tilde{f} \in SPF_m^n : f\Downarrow x \sqsubseteq \tilde{f} \wedge P(\tilde{f},x)$$

A specification is a commitment liveness condition for the function f if for every function with finite output there exists a function approximated by f that fulfills the specification.

Example: Commitment liveness condition
A simple commitment liveness condition is given by the following formula:

$$\exists z : \#f(x^\frown z) > 0$$

This condition expresses that for appropriately chosen continuations of finite input the output is not empty. □

An assumption specification P is called an *assumption liveness condition*, if

$$\forall x \in FIN[M^n] : \exists \tilde{x} \in (M^\omega)^n : x \sqsubseteq \tilde{x} \wedge P(f, \tilde{x})$$

A specification is an assumption liveness condition if for every finite input history x there exists an output history that is approximated by x and which fulfills the specification.

Example: Assumption liveness condition
A simple assumption liveness condition is given by the following formula.

$$\#x = \infty$$

This condition expresses that the input history is assumed to contain an infinite number of messages. □

Every safety specification $P_S \in ICS_m^n$ can be decomposed into a commitment safety condition P_S^C and an assumption safety condition P_S^E as follows:

$$P_S^C(f, x) \equiv \forall \bar{f} \in FIN[\downarrow f\Downarrow x] : \exists \tilde{f} : \bar{f} \sqsubseteq \tilde{f} \wedge P(\tilde{f}, x)$$

$$P_S^E(f, x) \equiv \forall \bar{x} \in FIN[\downarrow x] : \exists \tilde{x} : \bar{x} \sqsubseteq \tilde{x} \wedge P(f, \tilde{x})$$

Every liveness specification $P_L \in ICS_m^n$ can be decomposed into a commitment liveness condition P_L^C and an assumption liveness condition P_L^E as follows:

$$P_L^C(f, x) \equiv \exists \tilde{x} : P_S(f, \tilde{x}) \wedge x \sqsubseteq \tilde{x} \Rightarrow P_L(f, \tilde{x})$$

$$P_L^E(f, x) \equiv \exists \tilde{f} : P_S(\tilde{f}, x) \wedge f\Downarrow x \sqsubseteq \tilde{f} \Rightarrow P_L(\tilde{f}, x)$$

Then we obtain for a given specification $P(f, x)$ a more liberal component specification by

$$P_S^E(f, x) \Rightarrow P_S^C(f, x)$$

$$P_S^E(f, x) \ \wedge \ P_L^E(f, x) \Rightarrow P_L^C(f, x)$$

A given input choice specification Q can be decomposed into an assumption specification $A(f, x)$ defined by the following assertion

$$A(f, x) \equiv \exists \tilde{f}, \tilde{x} : x \sqsubseteq \tilde{x} \ \wedge \ f\Downarrow x \sqsubseteq \tilde{f} \ \wedge \ \neg Q(\tilde{f}, \tilde{x})$$

The commitment specification $C(f, x)$ then is defined by the following assertion

$$C(f, x) \equiv A(f, x) \ \wedge \ Q(f, x)$$

We obtain

$$Q(f, x) \equiv A(f, x) \Rightarrow C(f, x)$$

So far, we have studied the decomposition of input choice specifications from a more theoretical point of view. In the following section we take a more method-oriented view.

6 Assumption/Commitment Specifications

There are rather different ways and styles to write specifications of interactive components. In an assumption/commitment format a component is specified according to the following idea:

- if the environment fulfills certain assumptions, then the component fulfills its commitments.

Assumption/commitment specifications based on the idea of input choice specification are of the form

$$A(f,x) \Rightarrow C(f,x)$$

where A is an assumption specification, again called the *assumption* and C is a commitment specification again called the *commitment*.

In this section we introduce a general assumption/commitment format for input choice specifications. We first give examples for specifications written in this format and then describe the general format. We start by a simple example to explain a more sophisticated specification technique using the assumption/commitment format.

Example: One-element lossy buffer
A one-element lossy buffer is a component that may store at most one data element. It receives input messages which are either data elements or requests (represented by the signal Ⓡ). If the buffer never gets a request signal when it is empty and never gets a data message when it is full then it behaves properly like a one-element buffer, or it may be lossy. This means that it may lose a data message that is sent to it in an empty state. However, a message loss is indicated by the signal Ⓡ; if it stores a received data message correctly this is indicated by the signal $\surd$.

We specify the lossy bounded buffer of length 1 as follows. Recall the following definitions of message sets:

$$M = D \cup \{Ⓡ\}$$

$$N = D \cup \{Ⓡ\} \cup \{\surd\}$$

We specify the behavior of the component by functions

$$f : M^\omega \to N^\omega$$

with the help of the following four predicates:

Assumption safety:

$$P_S^E(f,x) \equiv \forall \tilde{x} \in M^* : \quad \begin{array}{l} \tilde{x}^\frown \langle d \rangle \sqsubseteq x \Rightarrow empty\,(\tilde{x}, f.\tilde{x}) \\ \tilde{x}^\frown \langle Ⓡ \rangle \sqsubseteq x \Rightarrow full\,(\tilde{x}, f.\tilde{x}) \end{array}$$

The predicates *empty* and *full* are specified as follows. For any pair (x, y) of streams where $x \in M^\omega$ is an input history and y is an output history the proposition $empty\,(x, y)$ indicates that the buffer is empty after having observed these histories. The proposition $full\,(x, y)$ indicates the same for the proposition that the buffer is full.

$$empty\,(x, y) = [\#D\textcircled{c}x \leq \#\{\textcircled{R}\}\textcircled{c}x + \#\{\textcircled{R}\}\textcircled{c}y]$$

$$full\,(x, y) = [\#D\textcircled{c}x > \#\{\textcircled{R}\}\textcircled{c}x + \#\{\textcircled{R}\}\textcircled{c}y]$$

Commitment safety:

$$P_S^C(f, x) \equiv \forall \tilde{x} \in FIN[\downarrow x] : h(\tilde{x}, f.\tilde{x})$$

Here h is an auxiliary predicate that specifies the required relationship between input and output. Formally this is expressed by the following equations:

$$h(d^\frown x, \textcircled{R}^\frown y) \equiv h(x, y)$$

$$h(d^\frown x, \surd^\frown y) \equiv \big(ft.x = \textcircled{R} \Rightarrow ft.y = d \ \wedge \ h(rt.x, rt.y)\big)$$

$$h(x, \langle\rangle) \equiv true$$

$$h(\langle\rangle, y) \equiv (y = \langle\rangle)$$

Assumption Liveness:

$$P_L^E(f, x) \equiv true$$

Commitment Liveness:

$$P_L^C(f, x) \equiv \#f.x \geq \#x$$

Since in all formulas used in the specification we just refer to $f.x$ and not to the function f in a more sophisticated way, it is obvious that we can also write the specification in the assumption/commitment format with prophecies as introduced in section 4.2. □

In the following section we develop a general assumption/commitment format for functional specifications.

7 General Assumption/Commitment Format

In this section we give a general assumption/commitment format for input choice specifications. It consists of an assumption

$$P^E(f, x)$$

and of a commitment predicate

$$P^C(f, x)$$

From these predicates we can derive safety and liveness specifications as follows

$$P_S^E(f,x) \equiv \forall z \in FIN[\downarrow x] : P^E(f,z)$$

$$P_S^C(f,x) \equiv \forall g \in FIN[\downarrow f] : P^C(g,x)$$

$$P_L^E(f,x) \equiv P_S^E(f,x) \Rightarrow P^E(f,x)$$

$$P_L^C(f,x) \equiv P_S^C(f,x) \Rightarrow P^C(f,x)$$

Using these predicates we can derive the specification for a component in the assumption/commitment format:

$$\begin{aligned} Q(f,x) \quad \equiv \quad & \big(P_S^E(f,x) \Rightarrow P_C^E(f,x)\big) \wedge \\ & \big(P_S^E(f,x) \ \wedge \ P_L^E(f,x) \Rightarrow P_L^C(f,x)\big) \end{aligned}$$

In many cases we do not need the concept of an input choice specification actually and can then write instead:

$$\begin{aligned} Q.f \quad \equiv \quad & \forall x : \big(P_S^E(f,x) \Rightarrow P_C^E(f,x)\big) \wedge \\ & \big(P_S^E(f,x) \ \wedge \ P_L^E(f,x) \Rightarrow P_L^C(f,x)\big) \end{aligned}$$

Finally we give an example of a specification written in this format.

Example: Unreliable one-element buffer
We consider a buffer that may store up to one element. It is unreliable in the sense, that it may ignore requests. It is fair in the sense that it does not ignore requests infinitely often. When it is full and it receives a request it may either send its content and become empty or it may send the signal Ⓡ indicating that it refuses to send its content and remains full. If it receives a data message when it is full or if it gets a request when it is empty, it breaks.

We define the set of messages M as follows:

$$M = D \cup \{Ⓡ\}$$

The functions

$$f : M^\omega \rightarrow M^\omega$$

with the required behavior are specified by the predicate Q which is defined by the following formula:

$$\begin{aligned} Q.f = \forall x \in M^* : \exists i \in I\!N : \quad & \big(\forall j \in I\!N : j \leq i \Rightarrow f(d^\frown Ⓡ^j) = Ⓡ^j\big) \wedge \\ & f(d^\frown Ⓡ^{i+1\frown} x) = Ⓡ^{i\frown} d^\frown f.x \end{aligned}$$

Here for a message m and $j \in I\!N$ we write m^j to denote the finite stream with exactly j occurrences of m.

Before we split the specification into assumptions and commitments, we give a general system invariant in terms of an input choice specification. This system invariant gives all the properties of the input streams and the corresponding output streams as long as the input fulfils the assumptions. This general system invariant $P^G(f,x)$ for any input stream x and the function f is formulated as follows:

$$\begin{array}{lll} P^G(f,x) \equiv & \#\big(\{\circledR\}\copyright x\big) = \#f.x & \wedge \\ & \#\{\circledR\}\copyright x \leq \#\{\circledR\}\copyright f.x + \#D\copyright x & \wedge \\ & \#D\copyright x \leq \#D\copyright f.x + 1 & \wedge \\ & D\copyright f.x \sqsubseteq D\copyright x & \wedge \\ & \exists j \in I\!N \cup \{\infty\} : \#\{\circledR\}\copyright x = j \Rightarrow D\copyright f.x = D\copyright x & \end{array}$$

Informally the five lines of the specification express the following properties of the buffer:

(1) The number of request signals in the input stream is identical to the number of elements in the output stream.

(2) The number of request signals in the input stream is less than the number of request signals in the output stream and the number of data elements in the input streams. In other words, there are as most as many requests in the input stream as data have been sent and requests have been rejected.

(3) There is at most one data message more in the input stream than in the output stream. In other words, at most one data element is stored in the buffer.

(4) The data in the stream produced by the buffer as output are a prefix of those received as input.

(5) If enough request signals are sent all data received as input are produced as output.

$P^G(f,x)$ includes both the requirements for the environment and for the component. However, since $P^G(f,x)$ also contains constrains for f there does not exist a function f such that

$$\forall x : P^G(f,x)$$

Therefore we split the predicate P^G into requirements for the component and those for its environment.

$$\begin{array}{lll} P^E(f,x) \equiv & \exists \tilde{x} : & \\ & \big(x = \tilde{x}\,\hat{}\,\circledR \wedge \#(D\copyright\tilde{x}) > \#(D\copyright f.\tilde{x})\big) & \vee \\ & \big(\exists d : x = \tilde{x}\,\hat{}\,d \wedge \#(D\copyright\tilde{x}) = \#(D\copyright f.\tilde{x})\big) & \end{array}$$

$$\begin{array}{lll} P^C(f,x) \equiv & D\copyright f.x \sqsubseteq D\copyright x & \wedge \\ & \#f.x = \#\{\circledR\}\copyright x & \wedge \\ & \exists j \in I\!N \cup \{\infty\} : \#\{\circledR\}\copyright x = j \Rightarrow D\copyright f.x = D\copyright x & \end{array}$$

The condition P^E is already nearly an assumption safety property. The component property P^C can be split into a commitment safety property P^C_S and a commitment liveness property P^C_L:

$$\begin{aligned} P^C_S(f,x) \equiv \quad & \forall \tilde{x} \in FIN[\downarrow x] : D\textcircled{c}f.\tilde{x} \sqsubseteq D\textcircled{c}\tilde{x} \quad \wedge \\ & \#f.\tilde{x} \leq \#\{\textcircled{R}\}\textcircled{c}\tilde{x} \end{aligned}$$

$$\begin{aligned} P^C_L(f,x) \equiv \quad & \#f.x \geq \#\{\textcircled{R}\}\textcircled{c}x \qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad \wedge \\ & \big(\exists j \in I\!N \cup \{\infty\} : \#\{\textcircled{R}\}\textcircled{c}x = j \Rightarrow D\textcircled{c}f.x = D\textcircled{c}x\big) \end{aligned}$$

This way we obtain a specification in an assumption/commitment format in the following form

Safety:

$$\big(\forall z : z \sqsubseteq x \Rightarrow P^E(f,z)\big) \Rightarrow P^C_S(f,x)$$

Liveness:

$$\big(\forall z : z \sqsubseteq x \Rightarrow P^E(f,z)\big) \Rightarrow P^C_L(f,x)$$

This example demonstrates already basically the most general assumption/commitment format. It also shows that the assumption/commitment format can be more involved than a straightforward functional specification. □

When giving a specification in the assumption/commitment format as used in the example above we write in the case of safety properties formulas of the form:

$$P^E_S(f,x) \Rightarrow P^C_S(f,x)$$

where P^E_S is an assumption safety property and P^C_S is a component safety property.

A "good" state of a computation then can be characterized by a function f and an input history x for which the following assertion holds:

$$\forall z : z \sqsubseteq x \Rightarrow P^E_S(f,z) \wedge P^C_S(f,z)$$

Clearly there are various different possibilities to formulate the safety assumption $P^E_S(f,x)$. For instance we may replace $P^E_S(f,x)$ by the formula

$$(1) \qquad \forall z : z \sqsubseteq x \Rightarrow P^E_S(f,z)$$

as well as by

$$(2) \qquad P^E_S(f,x) \vee \exists z : z \sqsubset x \wedge \neg P^C_S(f,z) \ \wedge \ P^E_S(f,z)$$

The formula (1) gives the strongest assumption and therefore the most liberal specification. If we use this assumption then for states that can only reached by incorrect component behaviors nothing is said about the further behavior of the component.

The formula (2) gives the most liberal assumption. It expresses that this more liberal assumption is fulfilled, if f does not fulfill the commitment for a proper prefix z of x which fulfills the assumption with respect to f.

8 Conclusion

Appropriate techniques for specifying the interface of system components are among the most important prerequisites for systems engineering. These techniques have to be modular if we want to carry out a strict top down development. Functional specification techniques are modular anyway as shown in [Broy 92]. Therefore for functional specifications modularity is not the motivation for an assumption/commitment format. In addition to modularity we require that specifications are well-structured, readable and, in particular, close to the user's conceptions. To achieve this, the assumption/commitment format for specifications is an interesting and promising candidate. It is quite obvious that in many situations the interface between a component and its environment should reflect certain assumptions about the environment (many components are only required to work properly if certain assumptions are fulfilled) and moreover the properties to which a component is committed under these assumptions.

As has been shown in the previous chapters there are a number of technical options when formalizing the assumption/commitment concept. Which of these options is superior can only be judged on the basis of consequent experimentation. For being able to deal with more sophisticated examples a good balance between proper formal foundations and pragmatic techniques for representing behaviors is required.

Functional specification techniques are modular, anyhow. The interest in an assumption/commitment format in the framework of functional system specification techniques is therefore not motivated by modularity, but by understandability, methodological adequacy and tractability.

Acknowledgement: Part of this work has been carried out during my stay at DIGITAL Systems Research Center. The excellent working environment and stimulating discussions with the colleagues at SRC, especially Jim Horning, Leslie Lamport, and Martin Abadi are gratefully acknowledged. I thank also Ketil Stølen who has given valuable comments on several versions.

References

[Abadi, Lamport 90] M. Abadi, L. Lamport: Composing Specifications. Digital Systems Research Center, SRC Report 66, October 1990

[deBakker et al. 90] J. W. de Bakker, W.-P. de Roever, G. Rozenberg (eds): Stepwise Refinement of Distributed Systems. Lecture Notes in Computer Science 430, Springer 1990

[Brock, Ackermann 81] J. D. Brock, W. B. Ackermann: Scenarios: A Model of Nondeterminate Computation. In: J. Diaz, I. Ramos (eds): Lecture Notes in Computer Science 107, Springer 1981, 225-259

[Broy 90] M. Broy: Functional Specification of Time Sensitive Communicating Systems. REX Workshop. In: [deBakker et al. 90], 153-179

[Broy 92] M. Broy: Compositional Refinement of Interactive Systems. Digital Systems Research Center, SRC Report 89, July 15, 1992

[Chandy, Misra 88] K. M. Chandy, J. Misra: Parallel Program Design: A Foundation. Addison Wesley 1988

[Dederichs, Weber 90] F. Dederichs, R. Weber: Safety and Liveness from a Methodological Point of View. Information Processing Letters, Vol. 36, No. 1, 1990, 25-30

[Jones 83] C. B. Jones: Specification and Design of (Parallel) Programs. In: R. E. A. Mason (ed): Information Processing 83, North Holland 1983, 321-332

[Jones 86] C. B. Jones: Systematic Program Development Using VDM. Prentice Hall 1986

[Lamport 83] L. Lamport: Specifying concurrent program modules. ACM Toplas 5:2, April 1983, 190-222

[Pandyá 90] P. K. Pandyá: Some Comments on the Assumption-Commitment Framework for Compositional Verification of Distributed Programs. In: [deBakker et al. 90], 622-640

[Stark 85] E. D. Stark: A Proof Technique for Rely/Guarantee Properties. Lecture Notes in Computer Science 206, Springer 1985

[Stølen 91] K. Stølen: Development of Parallel Programs on Shared Data Structures. Manchester University Technical Report, UNCS 1991-1-1.

[Stølen et al. 92] K. Stølen, F. Dederichs, R. Weber: Assumption/Commitment Rules for Networks of Agents. Manuscript 1992