Characterizing the Behavior of Reactive Systems by Trace Sets*

Manfred Broy
Institut für Informatik
Technische Universität München
Arcisstraße 21
D-80333 München

Revised and Extended Version of a Joint Paper with

Frank Dederichs, Claus Dendorfer, Rainer Weber

Abstract

The behavior of an asynchronous reactive system can be described by its set of action traces. In this paper, we investigate which properties a trace set fulfills describing a reactive system where input and output actions are distinguished. These properties reflect the fact that, for such a system, input actions can always occur. The required properties are discussed in the light of safety and liveness concepts and related to the concept of I/O-automata.

* This work was supported by the Sonderforschungsbereich 342 "Werkzeuge und Methoden für die Nutzung paralleler Rechnerarchitekturen" and the DFG-project "Transformation verteilter Systeme".

1. Introduction

Throughout this paper we use the term *reactive system* in the following sense (see [Harel, Pnueli 85]): a reactive system is an *open system* that is a system connected in some way to its environment that successively reacts to input stimuli issued by its *environment.* Accordingly a reactive system and its environment together form a *closed system.*

Numerous formal models representing the behavior of reactive systems have been proposed in computing science. Many of these proposals are based on the concept of *traces*, which are finite and infinite sequences of actions. When the *interface* of a reactive system is described, only those actions of the system and its environment are considered that may have some impact on the others behavior.

In the following we assume that all interface actions are uniquely classified as input or output actions. *Input actions* are performed by the environment. They are observed by the reactive system and influence its reactions. Vice versa, *output actions* are performed by the system and observed by the environment. Fig. 1 illustrates this view.

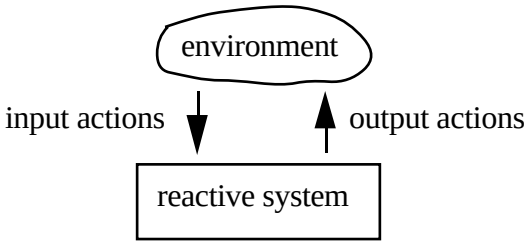

Fig. 1 Schematic representation of a reactive system

For a behavioral description of systems it is often useful to distinguish between *safety* and *liveness properties*. Safety properties refer to characteristics that are reflected by partial, finite observations, whereas liveness properties refer to characteristics of complete, possibly infinite observations. Infinite traces are mandatory to model liveness properties of reactive systems (cf. [Dill 89]).

The distinction between pure input and pure output actions imposes some requirements on trace sets that describe interfaces. Clearly, distinguishing between input and output actions and assuming asynchronous interaction implies that input actions are always enabled. This has to be reflected by the respective trace sets. Asynchronous reactive systems with I being the set of input actions and O being the set of output actions are called *I/O-systems* in the sequel. Informally speaking, in an I/O-system input actions are always enabled and therefore may always occur (at every position of a trace). Moreover, it should be possible for an I/O-system to

generate a correct trace no matter how the environment behaves, meaning independent of which patterns of input actions it chooses. Hence we assume that

(i) an I/O-system cannot influence which input actions occur and when,

(ii) at each point in a computation history (represented by a finite trace) an I/O-system does select its next output action based only on the information contained in the finite trace of input and output actions that occurred so far. Nevertheless, this selection may be done nondeterministically.

Assumption (ii) establishes a link to state machines or automata: if we understand a state as a representation of the relevant information about the finite computation history executed so far, then assumption (ii) is equivalent to the assumption that the respective trace is generated by a state machine. A trace set is called *realizable* by an I/O-system, if there is a system whose behavior (modeled in terms of input-output traces) corresponds to that trace set. In the following we characterize trace sets of reactive systems by referring to the above requirements (i) and (ii) only, without the introduction of an operational model (and not taking into account classical questions of computability either).

The paper is organized as follows: In **section 2.1** we discuss different characterizations for realizable trace sets and gradually come to stronger criteria leading to a hierarchy of characterizations.

The notion of a *strategy* (for a given trace set) introduced in **section 2.2** turns out to be central. We call a trace set *strategic*, if all its traces can be achieved using a strategy. Strategic sets have all the desired properties, thus we claim that strategies are an appropriate characterization of realizable sets.

Jonsson (cf. [Jonsson 85], [Jonsson 87]) and Lynch and Stark (cf. [Lynch, Stark 89]) suggest *I/O-automata* to characterize trace sets. I/O-automata are (possibly infinite) automata with transitions corresponding to input and output actions. With an I/O-automaton we associate the set of its accepted traces. In **section 3** we analyze I/O-automata with respect to our notion of realizability and assess their appropriateness.

In **section 4** we modify the notion of an I/O-automaton as well as the notion of a strategy. It is shown that the set of traces accepted by I/O-automata with *weak fairness* coincide with the trace sets generated by refined strategies. The proofs of various theorems are delegated to an **appendix**.

Our approach is related to that of [Lamport, Abadi 90]. There also strategies are considered as the key concept to realizability. The differences of their approach to our approach are as follows: first, we investigate why strategies are meaningful and how they capture the essentials stated above. Second, we do not aim at (trace) specifications for reactive systems (part of which may be realized) but at complete descriptions of the system behavior. Composition of descriptions is not considered either. However, for those interested in this question we refer to [Broy 93]. Third, we compare strategies with other notions proposed to capture the idea of reactive systems. Technically in contrast to Lamport's and Abadi's approach, our method is

based on actions and not on states. This difference, however, is not very significant, since there is a duality between state based descriptions and action trace based descriptions.

2. Descriptive and Operational Characterizations of Trace Sets

We already pointed out that a classification of actions into input and output actions introduces some "asymmetry" and "directedness" in the behavior descriptions of asynchronous reactive systems. This is reflected by a number of fundamental properties that can be observed for trace sets of I/O-systems. We start with "weak" straightforward properties and end up with useful characterizations of properties fulfilled by trace sets associated with I/O-systems. First, some basic notions will be introduced.

Throughout this paper we consider the two disjoint sets I and O of *input* and *output actions*, respectively. Let A be the set of all actions:

$$A =_{df} I \cup O$$

A *trace* of a reactive system is a finite or infinite sequence of actions. The set of all traces is denoted by A^ω.

$$A^\omega =_{df} A^* \cup A^\infty$$

Here, A^* stands for the finite and A^∞ for the infinite traces. Given $a \in A$ by $\langle a \rangle$ we denote the one element sequence. A *trace set* is a subset of A^ω. We now introduce some common operations on traces. Let $s, t \in A^\omega$; the concatenation of s and t is denoted by $s^\frown t$. If s is infinite, then $s^\frown t = s$. We write $s \sqsubseteq t$ if s is a prefix of t:

$$s \sqsubseteq t \Leftrightarrow_{df} \exists\, r \in A^\omega : s^\frown r = t.$$

The empty trace ε is the least element of the set of traces A^ω with respect to the prefix order. A set of traces $\{c_i \in A^\omega \mid i \in \mathbb{N}\}$ is called a *chain* , if $c_i \sqsubseteq c_{i+1}$ holds for all i. The least upper bound of a chain C always exists and is denoted by $\bigsqcup C$. Hence, A^ω is a complete partially ordered set. The length of a trace s is denoted by $\#s$; it is either a natural number or ∞, if s is infinite. For a given subset $B \subseteq A$, we write $B©s$ for the trace obtained from s by deleting all actions which are not in B. Formally:

$$B©\varepsilon = \varepsilon,$$

$$B©a^\frown s = a^\frown B©s \quad \text{if } a \in B,$$

$$B©a^\frown s = B©s \quad \text{if } a \notin B.$$

The interface (also called black box behavior) of an asynchronous reactive system can be modeled by a trace set $T \subseteq A^\omega$ where A is the set of input or output actions at the interface.

Every element $t \in T$ is called a *complete trace* of the system. Every finite trace $s \in A^*$ which is a prefix of some trace $t \in T$ is called a *partial trace* (for T). By $\downarrow T$ we denote the set of partial traces for T. The set of partial traces reflects all safety properties represented by T.

When decomposing a distributed system into a family of interacting components, we may distinguish the concept of synchronous interaction and that of asynchronous interaction. In synchronous interactions both the sender and the receiver must be ready for a communication to take place. Certain actions are carried out as shared actions of at least two components such as the reactive system and its environment. In asynchronous systems every action is carried out exclusively by one of the components, no matter whether or not a partner is ready for communication. Under the assumption of asynchrony we obtain only trace sets with particular properties.

2.1 Descriptive Characterizations

First we capture some aspects of I/O-systems by giving descriptive (i.e., non-operational) characterizations. We will use the notions "trace set" and "system" interchangeably, but of course we always mean "a system behavior described by a trace set" or "a trace set modeling the behavior of a system".

a) Local Safety and Liveness

As pointed out above we assume that an I/O-system does not restrict the selection of input actions. At any time (i.e. after every partial trace), the system must accept arbitrary input actions, and furthermore it must be able to complete every partial trace on its own, i.e., with output actions only: This property of the traces of I/O-systems is called *local safety, input enabledness and autonomous liveness*.

Definition 2.1 (*local safety, input enabledness and autonomous liveness*):

$$\textit{Local-SL}(T) \Leftrightarrow_{df} \varepsilon \in \downarrow T \wedge \forall r \in \downarrow T: (\forall i \in I: r\hat{}i \in \downarrow T) \wedge (\exists s \in O^{\omega}: r\hat{}s \in T) \qquad \square$$

At first glance one might expect that this characterisation already captures all necessary properties required for trace sets of I/O-systems. A locally safe, input enabled and autonomously live system is perfectly able to react to finite input. However, nothing is said about the proper handling of infinite input. In fact, such a system need not even accept infinitely many input elements, as is demonstrated by the following example:

Example 2.2: Let I, O and A be defined as above and let

$$T = \{t \in A^{\omega} \mid \#\, I \copyright t < \infty\}.$$

T is locally safe and live, but it does not allow an infinite stream of input actions. Here, the amount of data that can be sent by the environment is restricted. □

In the next example, we will see that a locally safe, input enabled and autonomously live system may constrain the environment's input rate.

Example 2.3: Let $I = \{i\}$, $O = \{o\}$. Consider the trace set

$$T = \{t \in A^\omega \mid \exists\, r \in A^*, s \in A^\omega : t = r\hat{}o\hat{}o\hat{}s\}.$$

It is easy to check that T is locally safe, input enabled and autonomously live: we may add an input action to every finite approximation of a trace in T, thus yielding again an approximation of a trace. Moreover, if the environment gives only a finite number of inputs the system is always able to extend the resulting trace such that an element of T is realized. Two successive copies of the output action o do the job. Considering infinite input, however, we observe a peculiar restriction. Although the environment may execute the action i at any time, it must eventually allow the system to perform two successive output actions. □

A behavior as given by the trace set of the example above may be regarded as including a restriction of the environment's input rate. Following the principle that an I/O-system does not restrict its environment's behavior we find it more adequate to avoid such restrictions and therefore strengthen our requirements for trace sets.

b) Predictive Liveness

Local safety, input enabledness and autonomous liveness do not sufficiently capture all our requirements for the treatment of infinite input. For a reactive system not only the sequence of environment actions is important, but also their relationship with the system's actions. Hence we define the notion of an *input modeling* which describes the input (actions) of the environment to the system, sliced into finite words. After each of the words the system may respond with a single output action. The refusal to produce a (proper) output action is modeled by the empty word. Conversely, the environment may allow several system outputs in a row by providing successively the empty input word. The asymmetry between the environment and the system is justified by the assumption that, from the system's point of view, the environment may deliver input at an arbitrarily high rate while there exists an upper bound for the response rate of the system.

Formally, an input modeling is a function $h : \mathbb{N} \to I^*$, where h is called *finite* if there exists an $n \in \mathbb{N}$ such that $h(j) = \varepsilon$ for all $j \geq n$. Otherwise, h is called *infinite*. Let the set of all input modelings $\mathbb{IM}$ be defined by

$$\mathbb{IM} =_{df} \mathbb{N} \to I^*.$$

Based on this definition we define what it means that a trace t corresponds to some input modeling h, denoted by $t \angle h$:

Definition 2.4 ($\angle$): Let $t \in A^\omega$, $h \in \mathbb{IM}$. We define:

$$t \angle h \Leftrightarrow_{df} \exists\, g \in (\mathbb{N} \to O \cup \{\varepsilon\}): t = \bigsqcup \{k(i) \mid i \in \mathbb{N}\}$$
where
$$k(0) = \varepsilon,\ k(i+1) = k(i)^\frown h(i)^\frown g(i)$$ □

Note that *Local-SL*(T) implies that, for any finite input modeling h, there exists a trace $t \in T$ such that $t \angle h$. However, this need not be the case for infinite input modelings. In both examples, 2.2 and 2.3, there is no trace t such that $t \angle h$ for the input modeling h with $h(j) = i$ for all $j \in \mathbb{N}$. The requirement of local safety and liveness as a characterization of trace sets of I/O-systems is incomplete: some infinite input modelings may be excluded by the trace set, and thus the behavior of the environment may be restricted.

Therefore, we strengthen the characterization of trace sets of I/O-systems by the requirement that the trace set should contain a trace for every infinite input modeling.

Definition 2.5 (*predictive liveness*):

$$\textit{Predictive-Live}(T) \Leftrightarrow_{df} \textit{Local-SL}(T) \wedge \forall\, h \in \mathbb{IM} : \exists\, t \in T : t \angle h$$ □

A predictive live system permits arbitrary infinite input (arbitrary infinite behavior of its environment). The above examples are not predictive live.

However, this straightforward combination of local and global requirements is too naive. It does not take into account that the system's response to infinite input has to be established step by step. At each step only a finite amount of the total input can be observed in order to determine the next output. The second conjunct of definition 2.5 just says that the appropriate output can be guaranteed if the total (future) input is known in advance; this is in contrast to a stepwise operational behavior, which requires that the final result is gradually approximated by the reactions to the finite initial parts of the input. To illustrate this point we give the following example.

Example 2.6: We consider the trace set

$$T = \{t \in A^\omega \mid \#(O \copyright t) = \infty \Leftrightarrow \#(I \copyright t) \neq \infty\}.$$

Here we have the paradoxical case that the system must react to finite input with infinite output, but infinite input must result in finite output. At each step the system is in a dilemma: if no more input is supplied it has to continue giving outputs forever. If the input will continue forever it eventually has to stop producing output elements. In order to make the right choice the system must predict the future. This is clearly in contrast to obvious operational characteristics. □

The example demonstrates that predictive liveness does not exclude trace sets where the system would have to have some knowledge of the future because its required behavior cannot be determined from the partial traces observed so far.

2.2 Operational Characterizations

The difficulties with the above characterizations suggest a closer look at the mode of operation of reactive systems. As a "gedankenexperiment" we consider the interaction between a system and its environment as a two player game, one player being the system, the other one being the environment. Given a trace set T, the I/O-system wins if the trace resulting from alternating moves of the players is an element of T. Otherwise, the environment wins. If the system is able to win (at least if it plays in an optimal manner) no matter which moves the environment does, we say that there exists a *strategy* for the trace set. Moreover, if every trace in T is a possible outcome of a game, where the system uses its strategy, we say that "T is realizable by that strategy". "Alternating moves" means that the environment may give some inputs (possibly none, but not infinitely many), then the system may issue at most one output, and so on. A strategy determines for the system what output action (if any) it should do next, depending only on the previous history. Thus it is guaranteed that the system's decisions do not take the future input into account.

a) Deterministic Strategies

First we think of a strategy as a mechanism which, in all situations, determines a unique next step of the system. We call such a strategy *deterministic*. A deterministic strategy can be modeled by a function $j: A^* \to O \cup \{\varepsilon\}$. The strategy function j takes the history leading to the current situation as input and yields the next move of the system. Let the set of all deterministic strategies be denoted by $\mathbb{S}$ where

$$\mathbb{S} =_{df} A^* \to O \cup \{\varepsilon\}.$$

For a given input modeling h and a strategy j, the behavior of a system in terms of a trace is uniquely determined. The expression ptrace(j, h, n) denotes this behavior up to the n-th step; it represents a "partial trace" of the system. The "total" (possibly infinite) behavior of a system with strategy j for an input modeling h is denoted by ctrace(j,h), and the set of all total traces of a strategy j is traces(j).

Definition 2.7 (*traces of a strategy*): Let $h \in \mathbb{IM}$ be an input modeling and $j \in \mathbb{S}$ be a strategy. We define the function:

$$\text{ptrace}: \mathbb{S} \times \mathbb{IM} \times \mathbb{N} \to A^*$$

that associates with an input modeling and a strategy the partial trace of the first $n+1$ moves of the environment and the corresponding moves of the system by the following definition:

$$ptrace(j, h, 0) =_{df} h(0),$$

$$ptrace(j, h, n+1) =_{df} ptrace(j, h, n)\hat{}j(ptrace(j, h, n))\ \hat{}h(n).$$

Based on this function we may associate an input modeling and a strategy with a complete trace by the function:

$$ctrace: \mathbb{S} \times \mathbb{IM} \rightarrow A^{\omega}.$$

It is specified by:

$$ctrace(j, h) =_{df} \bigsqcup \{ptrace(j, h, n) \mid n \in \mathbb{N}\}.$$

Based on this function we associate with a strategy a set of traces by the function:

$$traces: \mathbb{S} \rightarrow \wp(A^{\omega}).$$

It is specified by:

$$traces(j) =_{df} \{ctrace(j, h) \mid h \in \mathbb{IM}\}.$$ □

This definition implies that for all strategies $j \in \mathbb{S}$ and for all input modelings $h \in \mathbb{IM}$:

$$t \in ctrace(j, h) \Rightarrow t \angle h,$$

and therefore for all $j \in \mathbb{S}$:

$$\forall\, h \in \mathbb{IM}: \exists\, t \in traces(j): t \angle h.$$

This shows that strategies generate trace sets that deal with arbitrary input behaviors. The existence of a strategy is a possible characterization for requirements for trace sets.

A trace set is called *strategic* if it can be realized (generated) by a strategy. This is formalized by the following definition.

Definition 2.8 (*realizability by a strategy*):

$$Strategic(T) \Leftrightarrow_{df} \exists\, j \in \mathbb{S}: T = traces(j)$$ □

If T is strategic then every $t \in T$ is a possible run of the system/environment game, where the system behaves according to a fixed deterministic strategy. During this game every system move is determined only by the information available up to the current situation, and the final outcome $ctrace(j, h)$ only depends on the finite history fragments $ptrace(j, h, n)$. Note that the set given in example 2.6 is not strategic. This is because any strategy that attempts to realize T produces infinite output for finite input modelings. Since a player using a particular strategy never knows whether or not the input is finished, it eventually has to start producing its

outputs, although input still may arrive. Thus it will yield infinite output for infinite input, resulting in a trace which is not in T.

By our definition of strategies the output for a given partial trace is uniquely determined. Since reactive systems often exhibit nondeterministic behavior, we want to model nondeterminism on the level of trace sets, too. There is more than one way to do this.

b) Sets of Deterministic Strategies

First one may consider a set of strategies, each of which realizes a subset of T. The sets of traces generated by these strategies may well overlap and their union is required to be T. We define:

Definition 2.9 (*realizability by a set of strategies*):

$$\textit{Fully-Realizable}(T) \Leftrightarrow_{df} T = \cup\{Z \subseteq T : \textit{Strategic}(Z)\} \wedge T \neq \varnothing \qquad \square$$

We may imagine a system as a library of deterministic strategies. At the start of every game the system nondeterministically chooses one of these strategies. Once the choice is made the system sticks to this strategy for the entire game.

c) Nondeterministic Strategies

Another way to achieve nondeterminism is to allow strategies which, for every history, offer a choice of possible outputs, from which one is arbitrarily selected whenever the system makes a move. We call this generalization a *nondeterministic strategy*. A nondeterministic strategy can be seen as a mapping

$$j: A^* \rightarrow \wp(O \cup \{\varepsilon\}) \backslash \varnothing.$$

that determines a set of outcomes for every finite trace (including the decision to produce no output). The empty set is excluded such that the system has at least one choice (including the empty output ε). The set of all nondeterministic strategies is denoted by $\mathbb{NS}$.

Definition 2.10 (*traces of a nondeterministic strategy*): Let $h \in \mathbb{IM}$ be an input modeling and $j \in \mathbb{NS}$ be a nondeterministic strategy. The function

$$\text{ptrace-set} : \mathbb{NS} \times \mathbb{IM} \times \mathbb{N} \rightarrow \wp(A^*) \backslash \varnothing$$

associates the set of partial traces after j moves with an input modeling and a strategy:

$$\text{ptrace-set}(j, h, 0) =_{df} \{h(0)\},$$

$$\text{ptrace-set}(j, h, n+1) =_{df} \{x \hat{} y \hat{} h(n) \mid x \in \text{ptrace-set}(j, h, n) \wedge y \in j(x)\}.$$

Based on this definition we specify the function

$$\text{ctrace-set} : (\mathbb{NS} \times \mathbb{IM}) \to \wp(A^\omega)\backslash\varnothing$$

by:

$$\text{ctrace-set}(j, h) =_{df} \{ \bigsqcup \{t_n \mid n \in \mathbb{N}\} \mid \forall\, n \in \mathbb{N}: t_n \sqsubseteq t_{n+1} \wedge t_n \in \text{ptrace-set}(j, h, n) \}.$$

Finally we specify the function

$$\text{traces-set} : \mathbb{NS} \to \wp(A^\omega)\backslash\varnothing$$

by:

$$\text{traces-set}(j) =_{df} \cup\{\text{ctrace-set}(j, h) \mid h \in \mathbb{IM}\}.$$ □

The notion of realizability is defined as in the deterministic case:

Definition 2.11 (*realizable by a nondeterministic strategy*):

$$\textit{Strategic-ND}(T) =_{df} \exists\, j \in \mathbb{NS}: T = \text{traces-set}(j)$$ □

By now we have defined three different operational characterizations for trace sets that correspond to I/O-systems. In the remainder of this section we investigate their relationship and moreover relate them to the descriptive characterizations of the previous section.

It is obvious from the definitions that any trace set T which is realizable by a deterministic strategy is also realizable both by a nondeterministic strategy and by a set of strategies:

$$\textit{Strategic}(T) \Rightarrow \textit{Strategic-ND}(T)$$

$$\textit{Strategic}(T) \Rightarrow \textit{Fully-Realizable}(T)$$

However, the reverse directions do not hold in general, since nondeterministic systems cannot be realized by a single deterministic strategy. Consider the following example:

Example 2.12: Take an arbitrary set I, let $O = \{o\}$, and

$$T = \{t \in A^\omega \mid \exists\, r \in I^\omega : t = r \vee t = o\hat{}r\}.$$

Then T can be realized by the following nondeterministic strategy:

$$j(\varepsilon) =_{df} \{o, \varepsilon\}, \qquad j(r) =_{df} \{\varepsilon\} \quad \text{if } r \neq \varepsilon.$$

Alternatively, T is realized by the following set $J = \{j_1, j_2\}$ of deterministic strategies, which are defined by:

$$j_1(\varepsilon) =_{df} o, \qquad j_1(r) =_{df} \varepsilon \quad \text{if } r \neq \varepsilon,$$

$$j_2(r) =_{df} \varepsilon.$$

Obviously no single deterministic strategy can realize T entirely. □

The relation between nondeterministic strategies and sets of strategies is clarified by the following observation:

Observation 2.13: A nondeterministic strategy j_{nd} determines a set J of deterministic strategies such that traces-set(j_{nd}) = $\cup\{\text{traces}(j) \mid j \in J\}$, where J is obtained from j_{nd} by fixing all nondeterministic choices in advance. □

This implies that any trace set T realizable by a nondeterministic strategy is also realizable by a set of deterministic strategies:

$$\textit{Strategic-ND}(T) \Rightarrow \textit{Fully-Realizable}(T)$$

Again the reverse direction does not hold: there are trace sets that can be realized by a set of strategies but that cannot be realized by a single nondeterministic strategy:

Example 2.14: Let $I = \{i\}$, $O = \{o\}$, $T = \{t \in A^\omega \mid \#o©t < \infty\}$, $J = \{j_n \mid n \in \mathbb{N}\}$, where the single strategies j_n are defined as follows:

$$j_n(r) =_{df} o \qquad \text{if } \#o©r < n,$$

$$j_n(r) =_{df} \varepsilon \qquad \text{if } \#o©r \geq n.$$

Each strategy j_n realizes those traces which contain exactly n outputs, i.e.,

$$\text{traces}(j_n) = \{t \in A^\omega \mid \#o©t = n\}.$$

The union of all traces(j_n) obviously yields T, hence J (fully) realizes T. The system decides right at the beginning how many outputs it will perform. This is not so for a nondeterministic strategy: here, in every situation, the system has the choice to output an ε, or an o. Since the system can always decide to output an o it is not guaranteed that only finitely many o's occur. Therefore, T cannot be fully realized by a nondeterministic strategy. □

This fact, that nondeterministic strategies are less expressive than deterministic strategies is the reason for the use of so called prophecy variables in some specification and verification methods for reactive systems. We can also conclude from observation 2.13 that no additional expressive power is gained by considering sets of nondeterministic strategies.

The following lemma throws some light on the connection between descriptive and operational characterizations of trace sets:

Lemma 2.15: $\textit{Fully-Realizable}(T) \Rightarrow \textit{Predictive-Live}(T)$ □

See the appendix for a proof of this lemma. Definition 2.5 illustrates that there are trace sets which are predictive live but not fully realizable. Reviewing the results of this chapter we recognize that we have established a proper hierarchy of characterizations for trace sets as this is shown in Fig. 2:

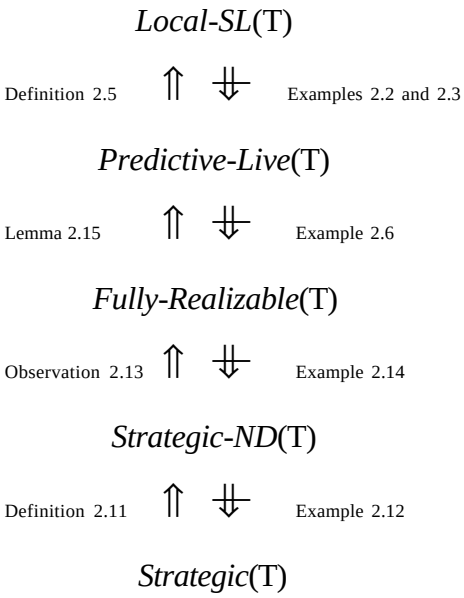

Fig. 2 Characterization Hierarchy for trace sets

We have seen that the first two notions only provide necessary conditions for the realization of a trace set T by a reactive system. Trace sets which only fulfill these conditions may either constrain the input of the environment or require unrealistic capabilities of the system. All this is avoided by the notion of full realizability. We claim that the concept of full realizability appropriately captures the operational behavior of a reactive system: it is as strong as necessary (*Local-SL* and *Predictive-Live* are not operational) and as weak as possible (neither *Strategic-ND* nor *Strategic* capture the full range of nondeterminism).

3. I/O-automata and their Accepted Words

Recently, I/O-automata have been suggested for the description of reactive systems. A definition is given in [Jonsson 87], and a similar definition can be found in [Lynch, Stark 89]. As usual in automata theory, I/O-automata can be considered a formal machinery for defining sets of traces, namely the sets of accepted words. Hence I/O-automata provide another way to characterize trace sets. In this section we review Jonsson's definitions of I/O-automata (using a slightly different notation), their *computations* and their *accepted traces.* We investigate how the corresponding trace sets fit into the hierarchy established in the preceding chapter.

An I/O-automaton communicates with its environment by (atomic) input and output actions. It meets a number of requirements, for example, it is always ready to accept arbitrary finite input from the environment (cf. [Jonsson 87]). The following definition formalizes this concept.

Definition 3.1 *(I/O-automaton)*: Consider quintuples $(I, O, \Sigma, \sigma_0, R, \mathbb{F})$, where:

- I is a set of *input actions*, not containing the silent action τ,
- O is a set of *output actions*, disjoint from I and not containing τ,
- Σ is a set of *states*,
- σ_0 is the *initial state* of the automaton,
- R is a subset of $\Sigma \times (I \cup O \cup \{\tau\}) \times \Sigma$, called the set of *labeled transitions*,
- $\mathbb{F}$ is a finite collection of *fairness sets*. Each fairness set is a subset of R.

We write $\sigma \xrightarrow{a} \sigma'$ to denote a labeled transition in R, where $\sigma, \sigma' \in \Sigma$ and $a \in (I \cup O \cup \{\tau\})$. An *I/O-automaton* is a tuple $(I, O, \Sigma, \sigma_0, R, \mathbb{F})$ which has the following properties:

a) For each state $\sigma \in \Sigma$ and input action $i \in I$ there is a state $\sigma' \in \Sigma$ such that $\sigma \xrightarrow{i} \sigma' \in R$.

b) No fairness set $F \in \mathbb{F}$ contains any transition $\sigma \xrightarrow{i} \sigma'$ for which $i \in I$.

c) Each transition $\sigma \xrightarrow{a} \sigma' \in R$ for which $a \in O \cup \{\tau\}$ is a member of some fairness set in $\mathbb{F}$. □

Jonsson uses silent transitions corresponding to internal steps of an automaton in order to model the composition of automata. In our context they are irrelevant; nevertheless we stick to his definition. The requirements a), b) and c) will be explained after the following two definitions:

Definition 3.2 (*transition enabling*): A transition $\sigma \xrightarrow{a} \sigma' \in R$ is called *enabled* in the state σ. A fairness set F is enabled in σ if a transition in F is enabled in σ. □

Definition 3.3 *(computations)*: A *computation* of an I/O-automaton N is a finite or infinite sequence of transitions

$$\sigma_0 \xrightarrow{a1} \sigma_1 \xrightarrow{a2} \ldots \xrightarrow{ak} \sigma_k \xrightarrow{ak+1} \ldots$$

which fulfills the following conditions:

1) *Initialization:* σ_0 is the initial state of N.

2) *State to state sequencing:* Each transition $\sigma_k \xrightarrow{ak+1} \sigma_{k+1}$ is a member of R.

3) *Strong Fairness:* If the sequence is infinite, then it contains infinitely many occurrences of transitions from each of those fairness sets $F \in \mathbb{F}$ which are enabled infinitely often.

4) *Quiescence:* If the sequence is finite, then no output transitions or silent transitions are enabled in its final state. □

Safety properties are expressed by 1) and 2). Conditions 3) and 4) are liveness properties. Together they form the criterion of acceptance for finite and infinite words (called *accepted traces*). Condition 3) expresses strong fairness as opposed to weak fairness, which would require that only transitions of those fairness sets are not ignored that, once enabled, remain enabled forever. We now explain the requirements a), b) and c) given in definition 3.1: requirement a) states that the automaton is always ready to accept inputs. b) says that fairness must not restrict the input, i.e., only the internal decisions of the automaton may be influenced by fairness considerations. Requirement c) expresses that the automaton is fair with respect to all silent and output transitions.

Definition 3.4 *(accepted traces)*: A *trace* of an I/O-automaton N is the sequence of input and output actions (that is, non-τ actions) in a computation of N. The set of all traces of N is denoted by traces(N). □

Jonsson shows in [Jonsson 87] that Kahn-networks [Kahn 74], and even nondeterministic Kahn-networks, can be modeled by a special class of I/O-automata. An I/O-automaton can also be used to implement strategies: given a nondeterministic strategy

$$j: A^* \to \wp(O \cup \{\varepsilon\})$$

we define the corresponding I/O-automaton $(I, O, A^*, \varepsilon, R, \mathbb{F})$ which describes the same trace set as j by:

$$(t, a, t') \in R \text{ iff } (a \in I \wedge t' = t\hat{}a) \vee (a \in O \wedge t' = t\hat{}a \wedge a \in j(t)),$$

$$\mathbb{F} = \{ R \cap (A^* \times (O \cup \{\tau\}) \times A^*) \}.$$

Here, of course, the choice of the fairness set is trivial.

Definition 3.5 *(realizable by an I/O-automaton)*: A trace set T is called *automatic* (and we write *Automatic*(T)) if there is an automaton N such that T = traces(N). □

We are now going to relate this characterization of trace sets to those defined in the previous chapter. Proofs that are omitted can be looked up in the appendix. First we consider the weakest property:

Lemma 3.6: *Automatic*(T) $\Rightarrow$ *Local-SL*(T) □

This result is not surprising. It simply states that I/O-automata appropriately handle enabledness for finite input. However, as the previous section shows, additional problems occur in connection with infinite input. Consider the following example, which is very much like the one used to illustrate the shortcomings of the local safe and live characterization.

Example 3.7: Let $I = \{i\}$, $O = \{o\}$, $T = \{t \in A^\omega \mid \exists\, r \in A^*, s \in I^\omega: t = r\hat{}o\hat{}o\hat{}s\}$. The reader is encouraged to check that the following I/O-automaton N has T as its set of accepted traces. Define the automaton $(I, O, \Sigma, \sigma_0, R, \mathbb{F})$ by

$$I = \{i\},\ O = \{o\},\ \Sigma = \{1, 2, 3\},\ \sigma_0 = 1,\ \mathbb{F} = \{\{1 \xrightarrow{o} 2\}, \{2 \xrightarrow{o} 3\}\},$$

and R as given by the transition diagram given in Figure 3:

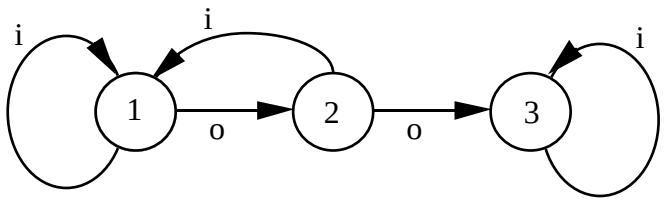

Fig. 3 Transition diagram

The trace set T requires that the environment eventually waits for two consecutive outputs to occur. This can be interpreted as a restriction on the input rate of the environment. □

The example also shows that *Automatic*(T) ⇒ *Predictive-Live*(T) does not hold, because the trace set form above is not predictive live. Also, the reverse direction is not valid as the following lemma demonstrates:

Lemma 3.8: The trace set $T = \{t \in A^\omega \mid \#O©t = \infty \Leftrightarrow \#I©t \neq \infty\}$ (cf. Example 2.6) is predictive live, but not automatic. □

From this one can easily infer that the fact that a trace set is locally safe and live does not ensure that it is automatic:

Corollary 3.9: *Local-SL*(T) ⇒ *Automatic*(T) does not hold. □

Hence the characterization of trace sets by I/O-automata is strictly stronger than the local safety and liveness characterization but cannot be compared with predictive liveness.

How is the relation between I/O-automata and strategies? It is apparent that these two characterizations are not equivalent: *Automatic*(T) ⇒ *Strategic*(T) cannot hold because of example 3.7 and by the same reason *Automatic*(T) ⇒ *Fully-Realizable*(T) cannot be true. However, we can prove the reverse directions:

Lemma 3.10: *Strategic*(T) ⇒ *Automatic*(T) □

Corollary 3.11: *Fully-Realizable*(T) ⇒ *Automatic*(T)

Proof: T can be realized by a set of strategies. By lemma 3.10, I/O-automata for each of these strategies exist. Building their disjoint union and identifying their initial states gives another I/O-automaton realizing T. □

Since a nondeterministic strategy corresponds to a set of deterministic strategies the above corollary immediately gives:

$$\textit{Strategic-ND}(\mathrm{T}) \Rightarrow \textit{Automatic}(\mathrm{T}).$$

Example 3.7 shows that I/O-automata may constrain their environment in some sense. Note that I/O-automata are defined with a *strong fairness* requirement, which means that, a fairness set which is enabled infinitely often does fire infinitely often. We might suspect that this requirement actually is too strong. An alternative fairness requirement is that of *weak fairness*, which means that only those fairness sets must fire infinitely often which are enabled continuously. The following proposition shows that the constraint of example 3.7 is indeed avoided when using weak fairness.

Lemma 3.12: For every I/O-automaton with weak fairness and at least one input action, there is a computation such that the corresponding trace does not contain two outputs in a row. □

All in all we established the following relationships:

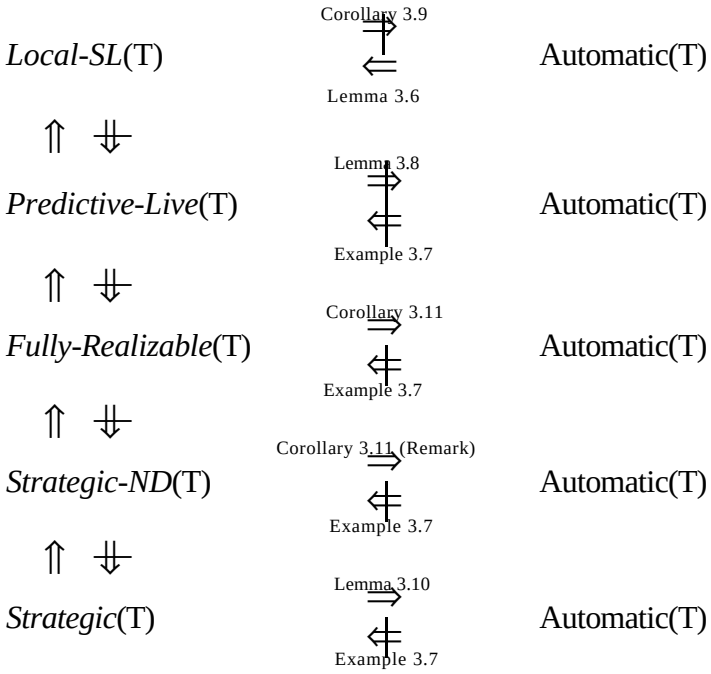

Fig. 4 Characterization hierarchy for trace sets and automata

The fact that *Automatic*(T) and *Fully-Realizable*(T) are not equivalent is rather unsatisfactory. Why should sets of strategies be less expressive than I/O-automata? An answer to this question is given in the following section. For a revised notion of a strategy an equivalence between being fully realizable and automatic with respect to weak fairness is established.

4. Weak Fairness and a Refined Notion of Strategies

At a first glance it seems that the requirement of strong fairness is the reason why the notions "*Automatic*" and "*Strategic*" do not coincide. Strong fairness may lead to restrictions of input rates (as demonstrated by example 3.7). Also for practical reasons such as the overhead required for guaranteeing strong fairness in implementations this concept has been considered as inadequate for models of reactive systems. Therefore we now switch to a notion of weak fairness. We replace the requirement of strong fairness by the following weaker notion.

Definition 4.1 (*Weak fairness, w.f.-automatic*): A computation sequence

$$\sigma_0 \xrightarrow{a1} \sigma_1 \xrightarrow{a2} \sigma_2 \xrightarrow{a3} \ldots \sigma_{k-1} \xrightarrow{ak} \sigma_k \xrightarrow{ak+1} \ldots$$

of an I/O-automaton N is called weakly fair, if the following condition holds: if a fairness set $F \in \mathbb{F}$ is continuously enabled, more formally, if for some $k \in \mathbb{N}$:

$$\forall\, i \in \mathbb{N}: k \le i \Rightarrow F \text{ is enabled in } \sigma_i ,$$

then the computation sequence contains infinitely many transitions from F.

We call a trace set T w(eakly) f(air) automatic, if T is accepted by an I/O-automata with weak fairness. We then write *Automatic-WF*(T). □

Our first conjecture, stated in a previous version of this paper

$$\textit{Automatic-WF}(T) \Rightarrow \textit{Strategic}(T)$$

was wrong as indicated by the following example due to Martin Abadi.

Example 4.2: Consider for the I/O-automaton with $I = \{i\}$, $O = \{o\}$ and $\mathbb{F} = \{\{1 \xrightarrow{o} 1\}, \{2 \xrightarrow{o} 2\}\}$ and R given by the state transition diagram given by Figure 5.

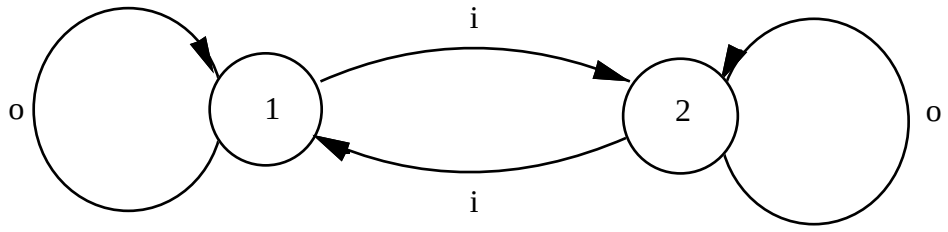

Fig. 5 Transition diagram

Under weak fairness which does not put any restriction here we obtain the following trace set T for this I/O-automaton:

$$T = (I \cup O)^{\infty}$$

However, T cannot be fully realized by any set of strategies. We prove this by contradiction. According to the results of the previous sections it is sufficient to consider a set J of deterministic strategies. Now assume that $T = \cup\{traces(j) : j \in J\}$. Then J cannot contain a strategy j such that

$$j(t) = \varepsilon$$

for some $t \in (I \cup O)^*$. This is true because otherwise t could be generated by j and a finite input modeling. Therefore t would be in T. This implies

$$j(t) = o$$

Therefore $i^\infty \notin T$ which is a contradiction. □

As the example demonstrates there are trace sets that are w.f.-automatic but not fully realizable. The reason is rather obvious: our notion of strategy is too simple. For a given strategy an empty input by the environment cannot be taken into consideration for determining the reactions of the I/O-system. This weakness is easily fixed.

Definition 4.3 (*Refined strategy*): A *refined strategy* is a function

$$j: (A^*)^* \to O \cup \{\varepsilon\}$$

The set of refined strategies is denoted by $\mathbb{RS}$. Again for every input modeling h a refined strategy defines a trace from A^* given by

$$flat(\bigsqcup \{rptrace(j, h, n) \mid n \in \mathbb{N}\})$$

where

$$rptrace: \mathbb{RS} \times \mathbb{IM} \times \mathbb{N} \to (A^*)^*$$

$$flat: (A^*)^* \to A^*$$

are specified as follows (note that elements of $(A^*)^*$ are sequences of sequences, let $\langle a \rangle \in (A^*)^*$ denote the one-element sequence just consisting of $a \in A^*$):

$$flat(\varepsilon) =_{df} \varepsilon$$

$$flat(\langle a \rangle) =_{df} a$$

$$flat(a^\frown b) =_{df} flat(a)^\frown flat(b)$$

and

$$rptrace(j, h, 0) =_{df} \langle h(0) \rangle$$

$$rptrace(j, h, n+1) =_{df} rptrace(j, h, n)^\frown \langle j(rptrace((j, h, n))) \rangle^\frown \langle h(n) \rangle$$

□

The notion of a refined strategy also allows us to obtain refined notions of a trace set being strategic, strategic-ND and fully realizable. We do not give these definitions explicitly, since they are rather straightforward, but concentrate on the question whether

$$RFR(T) \Leftrightarrow \textit{Automatic-WF}(T).$$

Here *RFR* stands for fully realizable by a set of refined deterministic strategies.
Obviously

$$\textit{Fully-Realizable}(T) \Rightarrow RFR(T)$$

because every strategy can be modeled by a refined strategy.
Before entering into the proof of the equivalence of RFR and *Automatic-WF* we give a definition that is helpful in the proof.

Definition 4.4 (*refined trace of an I/O-automaton*): An infinite sequence $s \in (A^*)^\infty$ is called a *refined trace* of an I/O-automaton M for input modeling h, if there is a trace t of M with $t \angle h$ and flat(s) = t. □

Now we have all definitions at hand to enter into the proof of our main theorem.

Theorem 4.5: For all trace sets T we have

$$RFR(T) \Leftrightarrow \textit{Automatic-WF}(T).$$

Proof: We show this equivalence by constructing for

(1) a given I/O-automata with weak fairness a set of refined deterministic strategies such that their trace sets coincide,
(2) a given set of refined deterministic strategies an I/O-automaton with weak fairness such that their trace sets coincide.

Add(1): Let the I/O-automaton $M = (I, O, \Sigma, \sigma_0, R, \mathbb{F})$ with weak fairness be given; for every input modeling h let

$$T_M(h) \subseteq (A^*)^\infty$$

denote the set of refined traces of the automaton M for the input modeling h.

Note that $T_M(h)$ is not empty, since M is input enabled and therefore for every input modeling we may construct a weakly fair computation by giving some weakly fair scheduling strategy. A simple strategy would be to select always one of those transitions in a fairness set which did not fire for the longest time span.

By the axiom of choice there exist functions

$$\eta: \mathbb{IM} \rightarrow (A^*)^\infty$$

such that

$$\eta(h) \in T_M(h)$$

Every function η defines a nondeterministic refined strategy:

$$j_\eta: (A^*)^* \to P(O \cup \{\varepsilon\})\backslash\{\emptyset\}$$

by

$$j_\eta(s) = \{o \in O \cup \{\varepsilon\} \mid \exists\, h:\ s\hat{}\langle o\rangle \in \downarrow\eta(h)\}$$

Again by the axiom of choice there exist deterministic strategies

$$\tilde{j}: (A^*)^* \to O \cup \{\varepsilon\}$$

with

$$\tilde{j}(s) \in j_\eta(s)$$

for all sequences $s, h \in \mathbb{IM}$. The set

$$J = \{\tilde{j}: (A^*)^* \to O \cup \{\varepsilon\} \mid \exists\, \eta:\ \mathbb{IM} \to (A^*)^\infty :$$

$$\forall\, h \in (A^*)^*: \eta(h) \in T_M(h) \wedge \forall\, s: \tilde{j}(s) \in j_\eta(s)\}$$

defines the set of all deterministic strategies with the required properties:

(a) Every $j \in J$ computes only traces that correspond to fair computations. This follows from the construction of $\tilde{j}$. Note s from $\tilde{j}(s) \in j_\eta(s)$ we have

$$t \in j_\eta(h) = \eta(h) \in T_M(h)$$

where

$$t = \bigsqcup \{rptrace(j_\eta, h, n) \mid n \in \mathbb{N}\}$$

(b) For any trace t we may do the choices of η and the strategy $\tilde{j}$ such that for the input modeling h corresponding to the computation the strategy $\tilde{j}$ we have

$$t = \bigsqcup \{rptrace(j_\eta, h, n) \mid n \in \mathbb{N}\}.$$

Add(2): Given a set J of deterministic refined strategies, the construction of an I/O-automata M with weak fairness with the same set of traces can be done as follows: define the automaton M by

$$M = \{I, O, \Sigma, \sigma_0, R, \mathbb{F}\}$$

Let σ_0 be an element that is not in J. We define

$$\Sigma = \sigma_0 \cup ((A^*)^* \times J) \quad \text{where } A = I \cup O$$

$$\sigma \xrightarrow{a} \sigma' \in R \text{ iff } (a \in A \cup \{\tau\} \wedge \sigma' \in res(\sigma,a))$$

$$\mathbb{F} = \{R \cap (\Sigma \times (O \cup \{\tau\}) \times \Sigma)\}$$

Here the function

$$\text{res}: \Sigma \times (A \cup \{\tau\}) \to \wp(\Sigma)$$

denotes resumptions for a given strategy j and a partial input trace and an action is defined by for $j \in J$, $s \in (A^*)^*$. Let $i \in I$, $o \in O$:

$$\text{res}([s,j],i) = \{[s\hat{}\langle i\rangle,j]\}$$

$$\text{res}(\sigma_0,i) = \{[\langle\langle i\rangle\rangle,j]: j \in J\}$$

$$\text{res}([s,j],o) = \{[s\hat{}\langle\langle o\rangle\rangle,j]: j(s) = o\}$$

$$\text{res}([s,j],\tau) = \{[s\hat{}\langle\varepsilon\rangle,j]: j(s) = \tau\}$$

$$\text{res}(\sigma_0,o) = \{[\langle\langle o\rangle\rangle,j]: j \in J\}$$

$$\text{res}(\sigma_0,\tau) = \{[\varepsilon,j]: j \in J\}$$

M is an I/O-automaton with a trivial fairness set where all nondeterministic choices (apart from input transitions) are done in the first step. It is a straightforward exercise to show that all the traces of M coincide with traces of the corresponding strategy. Note, the states in $\Sigma\backslash\{\sigma_0\}$ are pairs [t,j] which record the history of the computation in the form of the trace t and the (apart from input actions) deterministic future. □

The equivalence of *RFR* and *Automatic-WF* is a very satisfying result showing the close relationship between automata and strategies.

4. Conclusion

The work presented in the previous sections shows a close relationship to the paper by [Reingold et al. 91]. Since this paper contains a section comparing their work to a preliminary report of our work we simply refer to this section of their paper.

Although the work presented in the previous sections looks very much like a more sophisticated exercise in trace and automata theory it nevertheless contains a number of results that are also interesting from a more practical point of view. The close connection between trace sets for I/O-systems, sets of strategies and I/O-automata is certainly of practical value for the specification of asynchronous reactive systems. Particular results are that we can always work with trivial fairness sets (also observed in [Reingold et al. 91]) and that nondeterminism can always be represented by a set of deterministic strategies.

The separation of an action set into input and output actions in the modeling of asynchronous reactive systems by traces has a number of advantages:

(1) in many applications such a separation reflects the causalities for the actions more appropriately,

(2) simpler patterns of composition are available.

This leads to a more structured description of reactive systems. However, as shown in this paper, it also leads to a number of requirements for trace sets that are supposed to describe the interface of reactive systems. These requirements, however, give insight into operational concepts and their relationship to more abstract requirements such as fairness and even more general notions of liveness.

It has been pointed out be a number of researchers (cf. [Jonsson 87]) that there is a close relationship between I/O-automata and data flow networks. Data flow networks can also be modeled by sets of stream processing functions (cf. [Broy 89]). Note the similarity between a strategy and a stream processing function as well as between sets of strategies and sets of stream processing functions. It is an interesting observation that nondeterministic strategies can always be replaced by sets of deterministic strategies (but not vice versa). This shows that for asynchronous reactive systems we have a canonical separation of behaviors into (internal) nondeterminism and functional reaction to input.

Appendix

Proof of Lemma 2.15: Let T be a trace set such that *Fully-Realizable*(T) holds. Then there is a set J of deterministic strategies with $T = \cup\{traces(j) \mid j \in J\}$. We already know that for every strategy $j' \in \mathbb{S}$ and for every input modeling $h \in \mathbb{IM}$ there exists a $t \in traces(j')$ with $t \angle h$. In particular, for every $j \in J$ we have that $\forall\, h \in \mathbb{IM} : \exists\, t \in T : t \angle h$. This is the second conjunct of *Predictive-Live*(T).

It remains to be shown: *Fully-Realizable*(T) $\Rightarrow$ *Local-SL*(T).

Choose arbitrary $t \in T$, $i \in I$, and a finite $r \sqsubseteq t$. Since T is fully realizable there exists a $j \in J$ and an input modeling h such that $t = ctrace(j,h)$. Hence t has the form

$$t = h(0)\hat{}x(0)\hat{}h(1)\hat{}x(1)\hat{} \ldots$$

where $x(k) = j(h(0)\hat{}x(0)\hat{}h(1)\hat{}x(1)\hat{} \ldots \hat{}h(k))$ for all $0 \le k$. Since $r \sqsubseteq t$ there is an $n \in \mathbb{N}$, $n \ge 0$, where

$$r = h(0)\hat{}x(0)\hat{}h(1)\hat{}x(1)\hat{} \ldots \hat{}x(n-1)\hat{}y$$

with $x(i)$ as defined above and $y \sqsubseteq h(n)$. Now consider the following two input modelings:

$$h_1(k) = h(k) \qquad \text{if } k < n,$$

$$h_1(k) = y\hat{}i \qquad \text{if } k = n,$$

$h_1(k) = \varepsilon$ if $k > n$,

$h_2(k) = h(k)$ if $k < n$,

$h_2(k) = y$ if $k = n$,

$h_2(k) = \varepsilon$ if $k > n$.

Let $t_1 = ctrace(j, h_1)$ and $t_2 = ctrace(j, h_2)$. Clearly $t_1, t_2 \in traces(j) \subseteq T$. From the definition of ctrace we conclude: $ptrace(j,h_1,n) \sqsubseteq t_1$, $ptrace(j,h_2,n) \sqsubseteq t_2$.

Let $x_1(k) = j(h_1(0)\hat{}x(0)\hat{}h_1(1)\hat{}x(1)\hat{} \ldots \hat{}h_1(k))$. We calculate:

$$\begin{aligned} ptrace(j, h_1, n) &= h_1(0)\hat{}x(0)\hat{}h_1(1)\hat{}x(1)\hat{} \ldots \hat{}x(n-1)\hat{}h_1(n) \\ &= h(0)\hat{}x(0)\hat{}h(1)\hat{}x(1)\hat{} \ldots \hat{}x(n-1)\hat{}y\hat{}i \\ &= r\hat{}i \end{aligned}$$

Hence $r\hat{}i$ approximates the trace $t_1 \in T$. With respect to t_2 the same argument shows that $ptrace(j,h_2,n) = r$. Since $h_2(n+k) = \varepsilon$ for any $k>0$ it follows that $t_2 = r\hat{}s$ for an $s \in O^\omega$. This proves the second conjunct of *Local-SL*(T). □

Proof of Lemma 3.6: Given an I/O-automaton $N = (I, O, \Sigma, \sigma_0, R, \mathbb{F})$ and its accepted traces T, we have to show that:

$$\forall\, t \in T: \forall\, r \in A^*: r \sqsubseteq t \Rightarrow (\forall\, i \in I : \exists\, t' \in T: r\hat{}i \sqsubseteq t') \wedge (\exists\, s \in O^\omega: r\hat{}s \in T).$$

(1) First we show that $\forall\, t \in T : \forall\, r \in A^* : r \sqsubseteq t \Rightarrow (\forall\, i \in I : \exists\, t' \in T : r\hat{}i \sqsubseteq t')$. Let $t \in T$. Then there is a computation with accepted trace t. If $r \sqsubseteq t$ and r is infinite then we may choose $r = t = t'$. Otherwise r is finite and we consider the partial computation "up to r". This can be extended by an input transition with label i, because i is always enabled. The resulting partial computation can be extended with possibly infinitely many output or silent transitions until either no more output or silent transitions are enabled or the sequence becomes infinite. When extending the sequence at each step the transition is chosen from that fairness set which has the least number of transitions in the computation up to this moment. Thus the fairness requirement is fulfilled.

(2) It remains to be shown that $\forall\, t \in T: \forall\, r \in A^*: r \sqsubseteq t \Rightarrow (\exists\, s \in O^\omega: r\hat{}s \in T)$. Either r is already an accepted trace, or we can again extend it in a fair way with output and silent transitions as shown above. □

For the proof of Lemma 3.8 we need the following definition and lemma:

Definition 3.8.1 (*fairness count*): Given an I/O-automaton with fairness sets $F_1, \ldots, F_n$, we define the *fairness count* fc_i for each fairness set F_i as the function from partial computations to natural numbers where $fc_i(b)$ denotes how often F_i has been enabled since

its last firing in the partial computation b. The last state of the partial computation is not taken into account. □

So initially (for the empty computation) all fairness counts are equal to 0.

Lemma 3.8.2: Given an I/O-automaton with fairness sets $F_1,...,F_n$. We show that we can construct a chain of partial computations such that for every computation b in this chain there is a permutation $s: \{1,...,n\} \to \{1,...,n\}$ such that

$$\forall\, i : 1 \le i \le n \;\Rightarrow\; fc_{s(i)}(b) < i, \text{ and} \qquad (*)$$

b can be extended, if some fairness set is enabled after b. (**)

Proof: We prove the lemma by induction:

Base case: Let the permutation for the empty partial computation be any permutation. Obviously, (*) holds for this case.

Inductive case: Assume that (*) holds for a partial computation b, and let s be the associated permutation. If at least one fairness set is enabled in the last state of b, then we extend b by firing a transition from a fairness set $F_{s(k)}$, such that $k = \max\{i \in \{1,...,n\} \mid F_{s(i)} \text{ is enabled}\}$. Denote the extended computation by b', and define the new permutation s' as follows:

$s'(i) = s(i)$ if $i > k$,

$s'(i) = s(i-1)$ if $1 < i \le k$,

$s'(i) = s(k)$ if $i = 1$.

For $i > k$ we have:

$$fc_{s'(i)}(b') = fc_{s(i)}(b) < i.$$

For $1 < i \le k$ we have:

$fc_{s'(i)}(b') = fc_{s(i-1)}(b) + 1 < (i-1)+1 = i$ if $F_{s(i-1)}$ is enabled,

$fc_{s'(i)}(b') = fc_{s(i-1)}(b) < i-1 < i$ if $F_{s(i-1)}$ is not enabled.

For $i = 1$ we have:

$$fc_{s'(i)}(b') = fc_{s(k)}(b') = 0 < 1.$$

So $fc_{s'(i)}(b') = fc_{s(i)}(b) < i$ holds in every case; hence (*) is maintained. □

As a consequence of this lemma, all fairness counts of an I/O-automaton with fairness sets $F_1,...,F_n$ are bounded by n.

Proof of Lemma 3.8: We claim that the trace set $T = \{t \in A^\omega \mid \#O©t = \infty \Leftrightarrow \#I©t \neq \infty\}$ cannot be realized by any I/O-automaton. This is shown by proving that for every I/O-

automaton we can construct a computation (fulfilling the fairness and quiescence requirements) such that for the trace t of this computation either (1) or (2) hold:

(1) $\#I©t \neq \infty \wedge \#O©t \neq \infty$

(2) $\#I©t = \infty \wedge \#O©t = \infty$

Such a computation can be constructed step by step according to the following procedure:

We start with the initial state. Assume that we have already constructed a (finite) partial computation. There are two cases:

a) No τ- or O-transitions are enabled in its last state. Then this finite partial computation is already a computation and (1) holds for its trace t. In this case we stop the procedure.

b) Otherwise some fairness set is enabled, because the fairness sets contain all actions from $O \cup \{\tau\}$. We treat I approximately like a fairness set, but here we consider input transitions only *enabled* if, after the last firing of an input transition, an output transitions has fired. (Without this condition there can be firings of only τ- and I-transitions from some time on, resulting in a trace which contains infinitely many I-actions but only finitely many O-actions.) We extend this partial computation according to the rule given in the lemma above.

If this procedure is repeated infinitely often, then we have:

i) (1) or (2) holds.

ii) The partial computation is a computation, i.e., it fulfills the fairness requirement.

Consider i): If (2) does not hold, then there is a position in the partial computation after which either no more output transitions occur (i.e., only I- or τ-transitions occur) or no more input transitions occur (i.e., only O- and τ-transitions occur).

Consider the first case: this may happen if either:

α) infinitely many I-transitions occur or

β) finitely many I-transitions occur.

α) is also impossible, because then there must be two firings of input transitions with no firing of an output transition between them, contradicting the computation procedure. If β) holds we have case (1).

Consider the second case: this may happen if either:

α') infinitely many O-transitions occur or

β') finitely many O-transitions occur.

In the case of α') at least one fairness set fires infinitely often with O-transitions. But then, according to the computation procedure, also I-transitions occur infinitely often. If β') holds we have case (1).

Consider ii): If some fairness set was enabled infinitely often without firing, then its fairness count would become bigger than n, which is impossible because of the lemma above. □

Proof of Lemma 3.10: Since T is *Strategic*, we have T = traces(j) for some strategy j. Given j, we construct an I/O-automaton N = (I, O, Σ, σ_0, R, 𝔽) such that traces(N) = T.

Let I and O be as in j, and let $\Sigma = A^*$, $\sigma_0 = \varepsilon$. R is given by the following three kinds of transitions:

$w \xrightarrow{i} w\hat{}i$ for all $i \in I$:

$w \xrightarrow{o} w\hat{}o$ for $o \in O$, if $j(w) = o$,

$w \xrightarrow{\tau} w$, if $j(w) = \varepsilon$.

𝔽 contains the set of all non-input-transitions. We show that traces(j) = traces(N):

"⊆" : Let $t \in$ traces(j), i.e., t = ctrace(j, h) for some h. Then $t = h(0)\hat{}x(0)\hat{}h(1)\hat{}x(1)\hat{}...$ where $x(k) = j(\text{ptrace}(j, h, k))$ for all k and $\text{ptrace}(j,h,k) = h(0)\hat{}x(0)\hat{}...\hat{}h(k)$. We write $\sigma \xrightarrow{u} \sigma'$ for $u \in A^*$ to denote the extension of the transition relation to finite words. We obtain a computation

$$\varepsilon \xrightarrow{h(0)} h(0) \xrightarrow{act(x(0))} h(0)\hat{}x(0) \xrightarrow{h(1)} h(0)\hat{}x(0)\hat{}h(1) \quad ... \qquad (*)$$

where $act(\varepsilon) = \tau$, $act(o) = o$ for $o \in O$ ("doing nothing" is represented both by the action τ of an automaton and by the empty response ε of a strategy). This is a fair computation since we have only one fairness set.

"⊇": Let $t \in$ traces(N). Then there is an infinite computation which has trace t. The computation must be infinite, because j is a (total) function, such that in every state either τ or some $o \in O$ is enabled. The definition of the transition relation R implies that the computation must have the form (*) and furthermore $h(k) \in I^*$ for all k because of fairness. Thus $t \in$ traces(j). □

Proof of Lemma 3.12: We construct a computation that fulfills this condition. We start with the initial state. Assume we have already constructed a (finite) partial computation. Then we take an I-transition (which is always enabled). If no fairness set is enabled in the last state, then this finite partial computation is already a computation and fulfills our condition. Otherwise, we take a transition of a fairness set which is enabled and has a maximal fairness count with respect to. the other enabled fairness sets. Here, the *fairness count* of a fairness set says how often this fairness set has been enabled since its last firing (or the start of the computation). These steps are repeated until no fairness set is enabled. If there is always at least one enabled fairness set, then the partial computation will become infinite. Note that the (weak) fairness requirement is fulfilled. If this was not the case, then some fairness set F would eventually be enabled continuously without firing. For infinite computations, some other fairness set F' would have fired infinitely often. But then there would be a point where both F and F' are enabled and

the fairness count of F is bigger than that of F', which is not possible because of our procedure. □

References

[Broy 89]
M. Broy: A Functional Specification of Time-Sensitive Communicating Systems. ACM Transaction on Software Engineering and Methodology 2(1): pp. 1-46, 1993.

[Broy 93]
M. Broy: Towards a Design Methodology for Distributed Systems. In: Broy, M. (ed.): Constructive Methods in Computing Science. Springer, 1989, pp. 311 - 364

[Dill 89]
D.L. Dill: Trace Theory for Automatic Hierarchical Verification of Speed-independent Circuits. MIT Press, 1989

[Harel, Pnueli 85]
D. Harel, A. Pnueli: On the Development of Reactive Systems. In: K.R. Apt (ed.): Logic and Models of Concurrent Systems. Springer, 1985, pp. 477-498

[Hoare 85]
C.A.R. Hoare: Communicating Sequential Processes. Prentice Hall, 1985

[Kahn 74]
G. Kahn: The Semantics of a Simple Language for Parallel Programming. Information Processing 74, 1974, pp. 471-475

[Jonsson 85]
B. Jonsson: A Model and Proof System for Asynchronous Networks. Proc. 4th ACM Symposium on Principles of Distributed Computing, 1985, pp. 49-58

[Jonsson 87]
B. Jonsson: Compositional Verification of Distributed Systems. Ph.D. Thesis, Department of Computer Systems, Uppsala University, Uppsala, Sweden, DoCS 87/09, 1987

[Lamport, Abadi 90]
L. Lamport, M. Abadi: Composing Specifications. In: J.W. de Bakker, W.-P. de Roever, G. Rozenberg (eds.): Stepwise Refinement of Distributed Systems. LNCS 430, Springer 1990, pp. 1-41

[Lynch, Stark 89]
N. Lynch, E. Stark: A Proof of the Kahn Principle for Input/Output Automata. Information and Computation 82, 1989, pp. 81-92

[Reingold et al. 91]
N. Reingold, Da-Wei Wang, L.D. Zuck: Games I/O-Automata Play. Preliminary Report, Unpublished Manuscript, 1991