# On the realization of reactive systems*

Josep Carmona Jordi Cortadella

Software Department
Universitat Politècnica de Catalunya
Barcelona, Spain
{jcarmona,jordic}@lsi.upc.es

**Abstract**

A new notion of realization of reactive systems is defined. Realization is defined as a relation between the states of two transition systems, the specification and the implementation, in which events are classified as input, output or internal. This new definition attempts to model the correct interaction between a system and its environment. The differences with other definitions of refinement and realization are discussed.

## 1 Introduction

The concept of *realization* between a specification and an object implementing the specification's behavior has been approached by some authors [LT87, Jos88, BS95, Neg98, YG01]. A new definition of realization is defined here, in order to take into account the following facts:

1. The new realization notion must deal with *Reactive systems*, i.e., systems which react to changes of the environment. The environment where such systems operate can not be modified.
2. It is not necessary to maintain the overall concurrency between the events of the specification. Reducing the concurrency leads sometimes to simpler representations which are still correct realizations.

Basic definitions are given in Section 2. Section 3 summarizes some of the existing notions of realization and refinement. Section 4 is devoted to explain in detail the new notion of realization. An example is shown in Section 5.

## 2 Basic definitions

A Transition System (TS) [Arn94] is defined as a quadruple $A = (S, E, T, s_{in})$ where $S$ is a nonempty set of states, $E$ is a set of *events*, $T \subseteq S \times E \times S$ is the transition relation, and $s_{in} \in S$ is the initial state. The elements of $T$ are called the *transitions* of TS and are denoted by $(s, e, s')$ or $s \xrightarrow{e} s'$. A transition can also be denoted by $s \xrightarrow{e}$ if there is a state $s' \in S$ such that $s \xrightarrow{e} s'$.

The *reachability relation* between states is the transitive closure of the transition relation $T$. A *feasible sequence* is a (possibly empty) sequence of transitions $\sigma$ between states $s$ and $s'$ (denoted by $s \xrightarrow{\sigma} s'$). A *feasible trace* is obtained from a feasible sequence by removing states.

*This work has been partially funded by CICYT (TIC 98-0410 and TIC 98-0949), ACiD-WG (ESPRIT 21949), a grant by Intel Corporation, and CIRIT (1999SGR-150 and 2000FI-00472).

The set of events in $E$ is partitioned into input events ($E_I$), output events ($E_O$) and internal events ($E_{INT}$). The events in the sets $E_I$ and $E_O$ are called *observable* events.

A TS can be viewed as an automaton with alphabet $E$ where every state is an accepting state. For a TS $A$, let $L(A)$ be the corresponding language. We denote $w \setminus H$ the projection of $w$ to the alphabet $E - H$. Correspondingly, given a set of words $R$, the set $R \setminus H = \{w \setminus H \mid w \in R\}$.

# 3 Previous work

A summary of the existing relations between systems is done in this section. First, a well-known *equivalence* between systems is defined: the Observational Equivalence, defined in Milner's book [Mil89]. Afterwards two *refinement* notions are shown: $\sqsubseteq_J$, defined in [Jos88] and $\sqsubseteq_N$, defined in [Neg98]. Finally, two *realization* notions are shown: $\models_B$, defined in [BS95] and $\models_Y$, defined in [YG01].

## 3.1 Observational Equivalence

Several notions of equivalence have been defined in the literature. In Milner's work on theory of concurrent systems [Mil89], a notion of equivalence between two agents P and Q is defined.

**Observational Equivalence ($\approx$):** Let $A = (S, E, T, s_{in})$ and $B = (S', E', T', s'_{in})$ be two TSs. $A$ and $B$ are *observational equivalent* ($A \approx B$) iff there exists a relation $R \subseteq S \times S'$ satisfying

1. $(s_{in}, s'_{in}) \in R$.
2. (a) $\forall s \in S, \exists s' \in S'$ s.t. $(s, s') \in R$.
   (b) $\forall s' \in S', \exists s \in S$ s.t. $(s, s') \in R$.
3. (a) If $(s_1, s'_1) \in R$, $e \in (E_I \cup E_O)$ and $s_1 \xrightarrow{e} s_2$ then $\exists \sigma_1, \sigma_2 \in (E'_{INT})*$ such that $s'_1 \xrightarrow{\sigma_1 e \sigma_2} s'_2$, and $(s_2, s'_2) \in R$.
   (b) If $(s_1, s'_1) \in R$, $e \in (E_I \cup E_O)$ and $s'_1 \xrightarrow{e} s'_2$ then $\exists \sigma_1, \sigma_2 \in (E_{INT})*$ such that $s_1 \xrightarrow{\sigma_1 e \sigma_2} s_2$, and $(s_2, s'_2) \in R$.

The equivalence notion defined above is symmetric: the specification (implementation) can be either $A$ or $B$. According to our notion of realization, this is a rather strict relation because the same set of observable traces must be maintained in the implementation.

## 3.2 [Jos88]'s Refinement

In [Jos88] a set of simple rules was presented for the purpose of proving that a process $B$ can be shown to be a refinement of process $A$. In order to deal with internal events, the refinement with the *hiding operator*[1] will be defined here. It allows to have internal events in $B$:

**[Jos88]'s Refinement with hiding ($\sqsubseteq_J$):** Let $A = (S, E, T, s_{in})$, $B = (S', E', T, s'_{in})$ be two TSs, where $E'_I = E_I$, $E'_O = E_O$ and $E_{INT} = \emptyset$. $A \sqsubseteq_J B$ if there is a relation $D \subseteq S \times S'$ satisfying:

1. $\forall s_1 \in S, s'_1 \in S', e \in E : s_1 \xrightarrow{e} \wedge\, (s_1, s'_1) \in D \Longrightarrow \exists s'_2 \in S', \sigma \in (E'_{INT})^* : s'_1 \xrightarrow{\sigma e} s'_2$
2. (a) $\forall s_1 \in S, s_2, s'_2 \in S', e \in E'_{INT} : s_2 \xrightarrow{e} s'_2 \wedge\, (s_1, s_2) \in D \Longrightarrow (s_1, s'_2) \in D$
   (b) $\forall s_1 \in S, s_2, s'_2 \in S', e \in E : s_2 \xrightarrow{e} s'_2 \wedge (s_1, s_2) \in D \Longrightarrow \exists s'_1 \in S : s_1 \xrightarrow{e} s'_1 \wedge (s'_1, s'_2) \in D$
3. $(s_{in}, s'_{in}) \in D$

Refinement $\sqsubseteq_J$ does not distinguish the scope of the events (only events in $C$ are considered internal). Moreover, $A$ and $B$ must have the same set of observable events.

[1] This notion appears in the original paper as $P_1 \sqsubseteq P_2 \setminus C$, where $C$ is a set of internal events in $P_2$.

### 3.3 [Neg98]'s Refinement

Negulescu defines in [Neg98] the *Process Space Formalism*, a generic theory which includes, among others, the notion of process refinement. It regards the refinement as a relative notion of correctness: $B$ refines $A$ means that $A$ can be replaced by $B$ without bad effects. The notion of refinement is used there for verification purposes.

The theory of Process Spaces is briefly described here. Let $\mathcal{E}$ be an arbitrary set; the elements of $\mathcal{E}$ are called *executions*. A *process* over $\mathcal{E}$ is a pair $(X, Y)$ of subsets of $\mathcal{E}$ such that $X \cup Y = \mathcal{E}$. The set of all processes over $\mathcal{E}$ is called the *process space* of $\mathcal{E}$ and is denote by $\mathcal{S}_\mathcal{E}$.

The main intuition is the following: a process can describe a device by means of an *agreement* between the device and its environment, regarding executions. The agreement stipulates that only executions from $X \cap Y$ are allowed to occur in presence of the device. Set $\overline{X}$ contains executions where the device violates the agreement, while set $\overline{Y}$ contains the executions in which the environment violates the agreement. Accordingly, $X$ contains executions where the device respects the agreement (but the environment may or may not violate it), while $Y$ contains executions in which the environment respects the agreement (but the device may or may not violate it). For a process $p = (X, Y)$, set $X$ is called the *accessible* set of $p$ and set $Y$ is called the *acceptable* set of $p$.

The only parameter of a process space is the execution set; different models of processes can be choosed depending on what the executions represent. The notion of refinement in the Process Space formalism is quite simple:

**[Neg98]'s Refinement ($\sqsubseteq_N$):** Let $p, q \in \mathcal{S}_\mathcal{E}$ two processes. Process $q = (X_2, Y_2)$ refines process $p = (X_1, Y_1)$ ($p \sqsubseteq_N q$) iff $X_1 \supseteq X_2$ and $Y_1 \subseteq Y_2$.

Loosely speaking, when $p \sqsubseteq_N q$, $q$ accesses fewer (accepts more) executions than $p$. Therefore, depending on what is an execution, the refinement can be instantiated to several refinement notions between real systems. For instance, in the scope of reactive systems, an execution set can be $(E_I \cup E_O)^*$ or $(E_I \cup E_O \cup E_{INT})^*$. According to the definition of $\sqsubseteq_N$, no internal events can be inserted in $q$.

### 3.4 [BS95]'s Realization

Brzozowski and Seger [BS95] define a notion of realization between two behaviors: one denoting the specification's behavior and the other denoting the implementation's behavior.

Brzozowski and Seger's realization's notion is defined over the languages of the two TSs denoting specification's and the implementation's behavior. $A = (S, E, T, s_{in})$ and $B = (S', E', T', s'_{in})$ denote specification's and implementation's TS, respectively.

Let $\Sigma = \mathcal{P}(E_I \cup E_O)$ and $x \in \Sigma^*$ be a observable word in $L(A)$. A set $\mathcal{Y}$ of input variables, $\mathcal{Y} \subseteq E_I$, is *applicable* in A after $x$ if $\mathcal{Y} = \emptyset$ or if there exists a $\sigma \in \Sigma$ such that $x\sigma \in L(A) \setminus E_{INT}$ and $\sigma \cap E_I = \mathcal{Y}$. Word $w \in L(B) \setminus E'_{INT}$ is *relevant* to $A$ if $w \in L(A) \setminus E_{INT}$, or $w = x\sigma$, where $x \in L(A) \setminus E_{INT}$, $\sigma \in \Sigma$ and $\sigma \cap E_I$ is applicable to $A$ after $x$. Let $L(B/A)$ be the set of all the words of $B$ that are relevant to $A$. Loosely speaking, $L(B/A)$ contains those words in $B$ that matter when realizing $A$: suppose $x$ is input and $y$ is output and word $x_2 y_1 x_1 y_2$ is in $L(B)$, but no word in $L(A)$ allows $x_2$ to precede $x_1$. Then the word $x_2 y_1 x_1 y_2$ is not relevant to $A$.

Definitions of "deadlock" and livelock relative to the specification are also defined: let $w \in L(B) \cap L(A)$ be a observable word. $B$ has *deadlock* with respect to A if $w$ leads to a terminal state in $B$, but to a nonterminal state in $A$. Respectively, $B$ has a *livelock* with respect to $A$ if $w$ leads to a nonterminal state in $A$, and to a state in $B$ that has a cycle of internal events around it.

A TS is *input-proper* if, whenever $q_1 \xrightarrow{w'} q_2$, $q_1 \xrightarrow{w''} q'_2$, $x \in E_I$, $q_2 \xrightarrow{x} q_3$ and $w' \setminus E_{INT} = w'' \setminus E_{INT} = w$, then there also exists $q'_3$ such that $q'_2 \xrightarrow{x} q'_3$. In other words, whether or not an input change is permitted in a given state should depend only on the observable trace leading to that state.

**[BS95]'s Realization ($\models_B$):** Let $A = (S, E, T, s_{in})$ and $B = (S', E', T', s'_{in})$ be two TSs, with $E = E_I \cup E_O \cup E_{INT}$, $E' = E'_I \cup E'_O \cup E'_{INT}$, $E_I \subseteq E'_I$ and $E_O \subseteq E'_O$. $B$ realizes $A$ ($A \models_B B$) if

1. $A$ and $B$ are deterministic, and $B$ is input-proper.
2. $L(B/A) \setminus E_{INT} = L(A) \setminus E'_{INT}$.
3. $B$ is deadlock-free with respect to $A$.
4. $B$ is livelock-free with respect to $A$.

Restriction 2 requires that the set of relevant observable traces of the implementation must be equal to the set of observable traces of the specification. According to our intuition of realization, this is rather restrict for real life examples.

### 3.5 [YG01]'s Realization

Recently Yoeli and Ginzburg defined in [YG01] a new notion of realization over "Circuit Transition systems", which are TSs where a partition exists in the set of events, separating input, internal and output events. Let $A = (S, E, T, s_{in})$ and $B = (S', E', T', s'_{in})$ be two TSs, with $E = E_I \cup E_O$ and $E' = E'_I \cup E'_O \cup E'_{INT}$.

**[YG01]'s Realization ($\models_Y$):** $B$ realizes $A$ ($A \models_Y B$) iff

1. $E_I = E'_I$, $E_O = E'_O$.
2. $L(A) \subseteq L(B) \setminus E'_{INT}$
3. Assume $w \in L(A)$, $z \in E_O$, $w'z \in L(B)$ and $w' \setminus E'_{INT} = w$. Then $wz \in L(A)$.
4. Assume $w_1 w_2 \in L(A)$, and there exists $w' \in L(B)$, such that $w' \setminus E'_{INT} = w_1$. Then there exists $w''$, such that $w'' \setminus E'_{INT} = w_2$ and $w'w'' \in L(B)$.
5. Let $w \in L(A)$, $w' \in L(B)$, and $w' \setminus E'_{INT} = w$. Then there exists a positive integer $k$ such that for any word $w'' \in (E'_{INT})^*$, $w'w'' \in L(B)$ implies length($w''$) $< k$.

Two comments must be done in this definition: firstly, the input-proper restriction can be violated in the implementation. Secondly, Restriction 2 is rather restrict according to the notion of realization that we want to represent.

## 4 I/O preserving realization

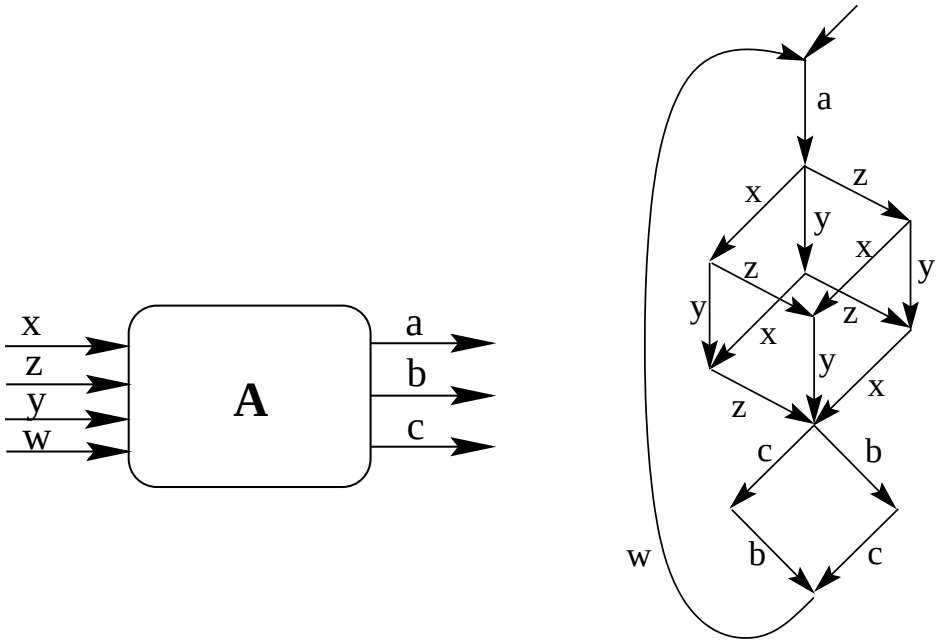
In this section the *I/O preserving realization* over reactive systems will be defined formally. It is inspired on the realization relation of Brzozowski and Seger, together with the refinement idea of Negulescu. We are interested in the combination of both approaches because the two relevant ideas presented in the introduction are represented: $\models_B$ express the input-proper restriction on the implementation, which contributes to preserve safely the dialog between the system and its environment. Secondly, Negulescu's idea of accessing the same or fewer executions represent the possibility of reducing the concurrency only between non-input events.

**Definition 4.1 (I/O preserving realization)**

Let $A = (S, E, T, s_{in})$ and $B = (S', E', T', s'_{in})$ be two TSs, with $E = E_I \cup E_O \cup E_{INT}$, $E' = E'_I \cup E'_O \cup E'_{INT}$, $E_I \subseteq E'_I$ and $E_O \subseteq E'_O$. $B$ realizes $A$ (A $\models$ B) iff there exists a relation $R \in S \times S'$ such that:

1. $(s_{in}, s'_{in}) \in R$.
2. If $(s_1, s'_1) \in R$ then:
   (a) $\forall i \in E_I$ s.t. $s_1 \xrightarrow{i} s_2$: $\exists s'_1 \xrightarrow{i} s'_2, \wedge\ (s_2, s'_2) \in R$.
   (b) $\forall o \in E_O$ s.t. $s'_1 \xrightarrow{o} s'_2$: $\exists s_1 \xrightarrow{o} s_2, \wedge\ (s_2, s'_2) \in R$.
3. $B$ is deadlock-free with respect to $A$.
4. $B$ is livelock-free with respect to $A$.

Condition 1 imposes that initial states must be related by $R$. Conditions 2(a)-(b) concern about the enabledness of the observable events both in the specification and the implementation.




(a) (b)

Figure 1: (a) Interface of system A, (b) Transition System model.

## 5 An example

The concepts will be presented with an example. Consider the system A, specified in Figure 1. It receives messages x, y, z and w, and generates messages a, b and c (Figure 1(a)). The behavior of such system is specified with the TS of Figure 1(b). Figure 2 depicts the TSs of four possible implementations of system A.

The notions $\approx$, $\sqsubseteq_J$, $\sqsubseteq_N$, $\models_B$, $\models_Y$ and the new realization notion $\models$ will be used for restricting the set of implementations of system A. The following table depicts the restricted search spaces. An "x" in cell (i,j) means that the element j belongs to the restricted state space when using relation i.

| | 2 (a) | 2 (b) | 2(c) | 2(d) |
|---|---|---|---|---|
| $\approx$ | x | x | | |
| $\sqsubseteq_N$ | | | x | |
| $\sqsubseteq_J$ | x | x | | |
| $\models_B$ | | x | | |
| $\models_Y$ | x | x | | |
| $\models$ | | **x** | **x** | |

Table 1: Restrictions of the implementation space.

According to the input-proper property, TS 2(a) can not be considered as a correct realization, due to the insertion of the internal event $\lambda$ before of the input messages x, y and z. The input-proper restriction can be explained with this example: imagine that the realization of $\lambda$ takes three days: 2(a) is imposing that after message a has been generated, the environment must wait for three days before of generating the messages x, y and z. Therefore, if implementation 2(a) was selected, the system(s) in the environment responsible of generating messages x, y and z must be resynthesized. The equivalence $\approx$, the realization $\models_Y$ and the refinement $\sqsubseteq_J$ include TS 2(a) in the restricted search space.

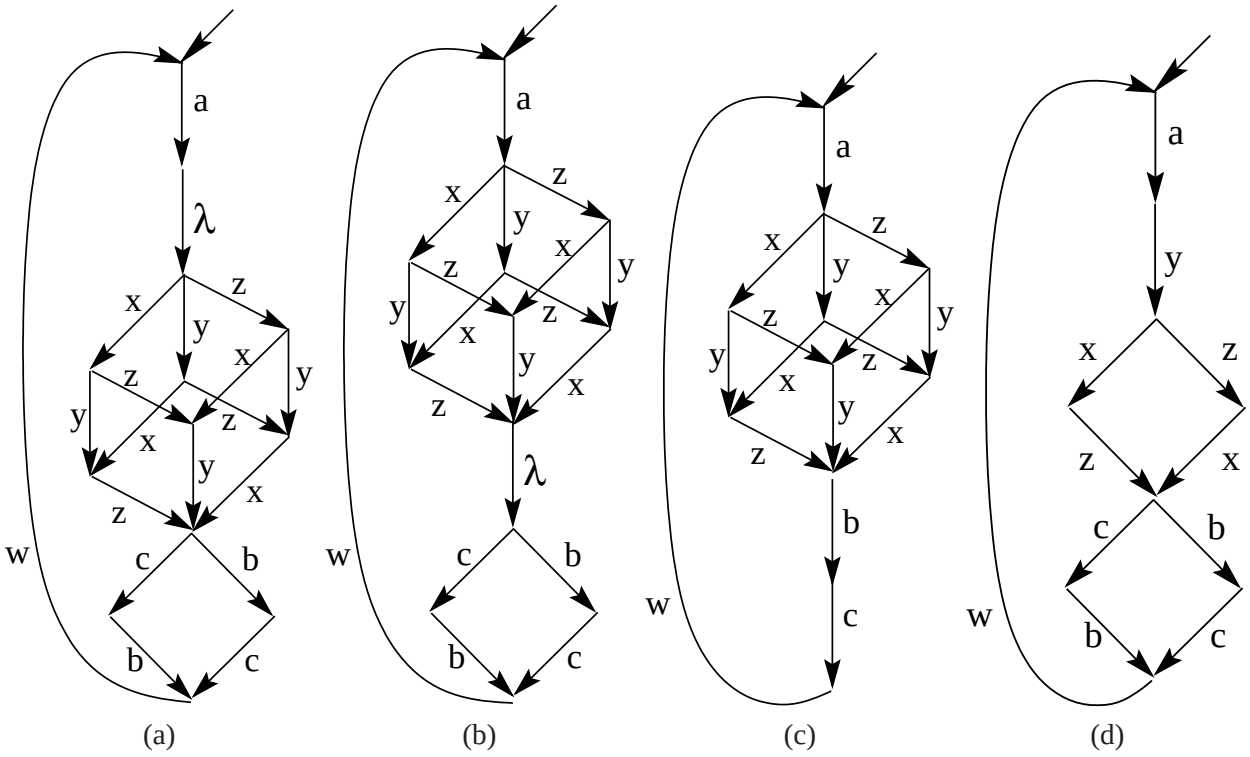



Figure 2: TS of different implementations of system A.

However, the insertion of the event $\lambda$ in 2(b) leads to an input-proper TS. Sometimes the specification must be modified in order to fulfill some implementability conditions. The transformations usually require the insertion of new internal events. The refinement $\sqsubseteq_N$ does not include TS 2(b) in the restricted search space.

According to our notion of realization, TS 2(c) is a correct implementation of system A, because it represents a modification of the specification 1(b), where the concurrency of the output events b and c is reduced: the environment will receive 2(c)'s messages in a more restricted (but still expected) order. Realization $\models_B$ does not include TS 2(c) in its restricted search space.

# 6 Conclusions

A new realization notion over reactive systems have been defined. It is based on the idea that a reactive system must operate in a distributed environment, where a fixed dialog exists between the system and the external systems (the environment). Comparison with respect to other definitions of equivalence, refinement and realization show the suitability of the new notion.

# References

[Arn94] A. Arnold. *Finite Transition Systems*. Prentice Hall, 1994.

[BS95] Janusz A. Brzozowski and Carl-Johan H. Seger. *Asynchronous Circuits*. Springer-Verlag, 1995.

[Jos88] Mark B. Josephs. A state-based approach to communicating processes. *Distributed Computing*, 3:9–18, 1988.

[LT87] Nancy A. Lynch and Mark R. Tuttle. Hierarchical correctness proofs for distributed algorithms. In *Sixth Annual ACM Symposium on Principles of Distributed Computing*, pages 137–151, Vancouver, British Columbia, Canada, August 1987.

[Mil89] Robin Milner. *Communication and Concurrency*. Prentice-Hall, 1989.

[Neg98] Radu Negulescu. *Process Spaces and Formal Verification of Asynchronous Circuits*. PhD thesis, Department of Computer Science, University of Waterloo, Waterloo, Ontario, Canada, August 1998.

[YG01] M. Yoeli and A. Ginzburg. Lotos/cadp-based verification of asynchronous circuits. Report CS-2001-09-2001, Technion - Computer Science Department, September 2001.