• 140000 8525
còpia 1

Acceptance automata:
a framework for specifying and verifying TCSP parallel systems

Luis M. Alonso
Ricardo Peña

Report LSI–90–44

Acceptance Automata: A Framework for Specifying and Verifying TCSP Parallel Systems

Luis M. Alonso
Departamento de Lenguajes y Sistemas Informáticos
Universidad del Pais Vasco
E-20080 San Sebastián
Spain.
email: alonso@gorria.if.ehu.es

Ricardo Peña
Departament de Llenguatges i Sistemes Informatics
Universitat Politecnica de Catalunya
E-08028 Barcelona
Spain.
email: ricardo@fib.upc.es

Abstract

Acceptance Automata, a particular case of labelled transition systems whose semantics is given in terms of the Failures Model, are presented. It is shown how parallel composition and hiding can be defined for them in a way consistent with the TCSP model. A notion of canonical automaton is presented. In the finite case it can be effectively computed and used for proving implementations correct. Finally, the TCSP concept of refinement is characterized in the acceptance automata domain.

1 Introduction

In a previous paper, see [PA 89], a technique for the specification, refinement and proof of correctness of parallel systems, was presented. The Failures Model [BHR 84][Hoa 85] was used as the mathematical model for Communicating Processes. Process behaviours were defined by means of a partial abstract type with certain characteristics. One of the advantages of this technique was that deductive methods developed in the general framework of algebraic specifications, could be used for proving properties concerning the specification of a process.

We call a refinement the decomposition of a process into a set of lower level processes, possibly abstracting the synchronization events. Proving that a refinement exhibits the same behaviour as the original process was considerably more difficult: a few and simple syntactic transformations were applied to the refinement. Such transformations were based upon the algebraic laws satisfied by the mathematical model. A similar verification technique is used in [HoJi 85]. Since then the style of specification has evolved, and we feel that it is powerful and general enough to be useful in most problems. Nevertheless, proving refinements correct by manipulation of algebraic expressions does not seem practical.

In what follows the algebra of Acceptance Automata is introduced. It provides another formal framework for communicating processes, while preserving the semantics defined by the Failures Model. Acceptance automata could be used in the definition of processes. However, their main interest is that they are amenable for mechanical manipulation and that verification techniques similar to bisimulations [Par 81] can be used. As it will be shown, there exists most often, a family of acceptance automata defining the same process. Nevertheless it will always be possible to choose a canonical and unique representative of a given family. When considering finite-state systems, the canonical form might be effectively computed to mechanically prove a refinement correct.

The idea of using transition systems to define processes in the Failures Model goes back to [Jos 88]. There are some differences with the approach taken here. In order to model non-determinism, there it was necessary to use several transitions, with the same original state and labelling event, and with different final states. This amounts to say that there is a family of transition systems defining the same process. The relationships among those transition systems are not clear enough. Moreover, none of them is marked as the canonical form, and it doesn't seem possible doing such a choice. Finally, in [Jos 88] a number of simple rules are provided, by means of which a process can be shown to be a refinement of another. Those rules are sufficient conditions but not general enough.

Another related approach is that of Acceptance Trees as described in [Hen 85][Hen 88]. The differences are mainly two: on the one hand we are interested in the parallel composition and hiding. These operations are not defined for Acceptance trees. On the other hand, we remain in the Failures Model, restricting ourselves to non-divergent processes.

Finally, a similar approach has been described in [AGS 90], where the so-called *CSP automata* are defined. They are similar to our acceptance automata, the main difference being the use of a *refusals mapping* in their definition. A notion of canonical form is also given. There are some minor differences: the work is restricted to finite-state systems and parallel composition and hiding are not defined. Besides that, the concept of refinement is not defined and different automata defining the same process are not compared.

2. Acceptance Automata

Definition 2.1 An *Acceptance Automaton* is given by a tuple $(\Lambda, \Sigma, \iota, \mathrm{M}, \rightarrow)$ where:

- Λ is a non-empty alphabet of events,
- Σ is the set of states
- $\iota \in \Sigma$, is *the* initial state
- $M \subseteq \Sigma \times \mathcal{P}(\Lambda)$, is a menu relation, that will be denoted: $m \in M(\sigma)$
- $\rightarrow \subseteq \Sigma \times \Lambda \times \Sigma$, is a transition relation, that will be denoted: $\sigma \rightarrow_e \sigma'$

M and $\rightarrow$ shall be defined so that:

1) M is total in its domain, i.e.: $\forall \sigma \in \Sigma, \exists m \in \mathcal{P}(\Lambda).\ m \in M(\sigma)$
2) $\forall \sigma \in \Sigma, m \in M(\sigma), e \in \Lambda.\ e \in m \Rightarrow (\exists \sigma' \in \Sigma.\ \sigma \rightarrow_e \sigma')$
3) $\forall \sigma, \sigma' \in \Sigma, e \in \Lambda.\ \sigma \rightarrow_e \sigma' \Rightarrow (\exists m \in M(\sigma).\ e \in m)$

The role of the menu relation is to represent the internal non-determinism in the system, whilst the states represent the past of the system. We want to remark in this point, that the initial state is unique. Moreover, the transition systems defined in [Jos 88] can be translated into acceptance automata, provided that their initial states are unique. In particular, it is possible to define an acceptance automaton with several transitions, with the same original state and labelling event, and with different final states. The reason for which we allow this situation will be made clear later in the paper. Nevertheless, we will say in advance that acceptance automata used in the definition of processes do not exhibit this characteristic.

Two examples of acceptance automata follow: the first one is a finite state system, while the second one is a system with an infinite number of states:

$\Lambda = \{a, b, c\}$
$\Sigma = \{0, 1\}$
$\iota = 0$
$M = \{(0, \{a\}), (1, \{b\}), (1, \{c\})\}$
$\rightarrow = \{(0, a, 1), (1, b, 0), (1, c, 0)\}$

$\Lambda = \{in, out\}$
$\Sigma = \mathbb{N}$
$\iota = 0$
$M = \{(i, \{in\}) \mid i \in \mathbb{N}\} \cup \{(i, \{in, out\}) \mid i \in \mathbb{N}^+\}$
$\rightarrow = \{(i, in, i+1) \mid i \in \mathbb{N}\} \cup \{(i, out, i - 1) \mid i \in \mathbb{N}^+\}$

The following definitions introduce the concepts of traces and reachable states in this context. After that we will define the failures semantics of an acceptance automaton. In what follows we assume an acceptance automaton $\mathcal{A} = (\Lambda, \Sigma, \iota, M, \rightarrow)$.

Definition 2.2 The *reflexive-transitive closure* $\rightarrow \subseteq \Sigma \times \Lambda^* \times \Sigma$ of the transition relation is defined as the smallest relation satisfying:

- $\forall \sigma \in \Sigma.\ \sigma \rightarrow_{<>} \sigma$
- $\forall \sigma_1, \sigma_2, \sigma \in \Sigma, t \in \Lambda^*, e \in \Lambda.\ \sigma_1 \rightarrow_t \sigma \wedge \sigma \rightarrow_e \sigma_2 \Rightarrow \sigma_1 \rightarrow_{te} \sigma_2$

<u>**Definition 2.3**</u> *traces*($\mathcal{A}$) is the set defined by: traces($\mathcal{A}$) $= \{t \in \Lambda^*.\ \exists\, \sigma \in \Sigma.\ \iota \rightarrow_t \sigma\}$

<u>**Definition 2.4**</u> The *reachability set of* $\mathcal{A}$ *after the trace t* is the set defined by:

$\mathcal{A}/t = \{\sigma \in \Sigma.\ \iota \rightarrow_t \sigma\}$

<u>**Definition 2.5**</u> The *reachability set of* $\mathcal{A}$ is the set defined by:

reach($\mathcal{A}$) $= \cup_{t \in \text{traces}(\mathcal{A})} \mathcal{A}/t$

<u>**Definition 2.6**</u> The *set of next possible events*, for every state $\sigma \in \Sigma$, is the set defined by:

$\text{next}(\sigma) = \{e \in \Lambda.\ \exists\, \sigma' \in \Sigma.\ \sigma \rightarrow_e \sigma'\}$

<u>**Definition 2.7**</u> The *failures semantics* of $\mathcal{A}$ is given by a relation $\mathcal{F}(\mathcal{A}) \subseteq \Lambda^* \times \mathcal{P}(\Lambda)$ defined as the smallest relation satisfying:

1) $\forall t \in$ traces $(\mathcal{A}), \sigma \in \mathcal{A}/t, m \in M(\sigma).\ (t, \neg m) \in \mathcal{F}(\mathcal{A})$
2) $\forall t \in$ traces $(\mathcal{A}), F_1, F_2 \in \mathcal{P}(\Lambda).\ (t, F_1) \in \mathcal{F}(\mathcal{A}) \wedge F_2 \subseteq F_1 \Rightarrow (t, F_2) \in \mathcal{F}(\mathcal{A})$

<u>**Lemma 2.8**</u> Given an acceptance automaton $\mathcal{A} = (\Lambda, \Sigma, \iota, M, \rightarrow)$, the triple $(\Lambda, \mathcal{F}(\mathcal{A}), \emptyset)$ defines a non-divergent process in the failures model. In what follows, we simply use $\mathcal{F}(\mathcal{A})$ to denote this process.

<u>Proof</u> We only have to show that the following conditions hold [Hoa 85]:

- $(<>, \emptyset) \in \mathcal{F}(\mathcal{A})$
  this is trivial from condition 1 in definition 2.7
- $(st, X) \in \mathcal{F}(\mathcal{A}) \Rightarrow (s, \emptyset) \in \mathcal{F}(\mathcal{A})$
  this comes from the fact that traces($\mathcal{A}$) is prefix closed, hence if $(st, X) \in \mathcal{F}(\mathcal{A})$ then $s \in$ traces $(\mathcal{A})$ and so $(s, \emptyset) \in \mathcal{F}(\mathcal{A})$
- $(s, Y) \in \mathcal{F}(\mathcal{A}) \wedge X \subseteq Y \Rightarrow (s, X) \in \mathcal{F}(\mathcal{A})$
  this is trivial from condition 2 in definition 2.7,
- $(s, X) \in \mathcal{F}(\mathcal{A}) \wedge x \in \Lambda \Rightarrow (s, X \cup \{x\}) \in \mathcal{F}(\mathcal{A}) \vee (sx, \emptyset) \in \mathcal{F}(\mathcal{A})$
  this comes from the fact that, s being a trace of $\mathcal{A}$, there exists state $\sigma \in \mathcal{A}/s$, and from the fact that, (s, X) being a failure, there exists a menu $m \in M(\sigma)$ such that $X \subseteq \neg m$. Hence, from definition 2.7, if $x \notin m$, then $(s, X \cup \{x\}) \in \mathcal{F}(\mathcal{A})$. Otherwise, if $x \in m$ then from condition 2 in definition 2.1 there exists a transition $\sigma \rightarrow_x \sigma'$ so sx is a trace

of $\mathcal{A}$ and $(sx, \emptyset) \in \mathcal{F}(\mathcal{A})$. ◊

This lemma implies that divergent processes can not be modelled by means of acceptance automata. We have taken this decission because we find no practical interest in specifying processes which are allowed to diverge.

The following definition introduces a notion of equivalence between acceptance automata given in semantic terms. Later on, this concept will be characterized in terms closely related to bisimulations defined between transition systems.

Definition 2.9 Given acceptance automata $\mathcal{A}_1$ and $\mathcal{A}_2$, they are said to be *observationally equivalent*, denoted by $\mathcal{A}_1 \approx \mathcal{A}_2$, if $\mathcal{F}(\mathcal{A}_1) = \mathcal{F}(\mathcal{A}_2)$, i.e. if they have the same failures semantics.

See in the following example two equivalent acceptance automata:

$\Lambda = \{\text{tick}\}$
$\Sigma = \{\text{"on"}\}$
$\iota = \text{"on"}$
$M = \{(\text{"on"}, \{\text{tick}\})\}$
$\rightarrow = \{(\text{"on"}, \text{tick}, \text{"on"})\}$

$\Lambda = \{\text{tick}\}$
$\Sigma = \mathbb{N}$
$\iota = 0$
$M = \{(i, \{\text{tick}\}) \mid i \in \mathbb{N}\}$
$\rightarrow = \{(i, \text{tick}, i+1) \mid i \in \mathbb{N}\}$

Up to this moment, the basic notions related with acceptance automata have been established. We proceed now to define the basic operations over acceptance automata. These operations shall correspond to the parallel composition and hiding defined in the Failures Model. None of these operators is defined for acceptance trees defined in [Hen 85], [Hen 88]. Whilst the parallel composition of acceptance trees seems to be rather simple, we think that the definition of the hiding operator will not be so simple if we try to do it strictly in the acceptance tree domain.

Definition 2.10 Given acceptance automata $\mathcal{A}_1 = (\Lambda \cup \Lambda_1, \Sigma_1, \iota_1, M_1, \rightarrow_1)$ and $\mathcal{A}_2 = (\Lambda \cup \Lambda_2, \Sigma_2, \iota_2, M_2, \rightarrow_2)$, with $\Lambda, \Lambda_1, \Lambda_2$ pairwise disjoint, the *parallel composition* of $\mathcal{A}_1$ and $\mathcal{A}_2$, denoted by $\mathcal{A}_1 \parallel \mathcal{A}_2 = (\Lambda_\parallel, \Sigma, \iota, M, \rightarrow)$, has Λ as synchronization alphabet and is defined as follows:

1) $\Lambda_\parallel = \Lambda \cup \Lambda_1 \cup \Lambda_2$
2) $\Sigma = \Sigma_1 \times \Sigma_2$,

3) $\iota = (\iota_1, \iota_2)$,

4) $M \subseteq \Sigma \times \mathcal{P}(\Lambda)$, is the smallest relation satisfying:

$\forall\, m_1 \in M_1(\sigma_1), m_2 \in M_2(\sigma_2)$:

$$(m_1 \cap m_2) \cup (\Lambda_1 \cap m_1) \cup (\Lambda_2 \cap m_2) \in M((\sigma_1, \sigma_2))$$

5) $\rightarrow \subseteq \Sigma \times \Lambda \times \Sigma$, is the smallest relation satisfying:

$\forall\, e \in \Lambda, \sigma_1 \rightarrow_{1_e} \sigma_2, \mu_1 \rightarrow_{2_e} \mu_2.\ (\sigma_1, \mu_1) \rightarrow_e (\sigma_2, \mu_2)$

$\forall\, e \in \Lambda_1, \sigma_1 \rightarrow_{1_e} \sigma_2, \mu \in \Sigma_2.\ (\sigma_1, \mu) \rightarrow_e (\sigma_2, \mu)$

$\forall\, e \in \Lambda_2, \sigma \in \Sigma_1, \mu_1 \rightarrow_{2_e} \mu_2.\ (\sigma, \mu_1) \rightarrow_e (\sigma, \mu_2)$

Fact 2.11 The parallel composition of acceptance automata $\mathcal{A}_1$ and $\mathcal{A}_2$, is another acceptance automaton $\mathcal{A}$.

Fact 2.12 Observe that the following predicate holds for acceptance automata $\mathcal{A}_1$ and $\mathcal{A}_2$, and its parallel composition $\mathcal{A}$:

$\forall\, t_1 \in \text{traces}(\mathcal{A}_1), \sigma_1 \in \mathcal{A}_1/t_1, t_2 \in \text{traces}(\mathcal{A}_2), \sigma_2 \in \mathcal{A}_2/t_2, t \in \text{traces}(\mathcal{A})$.

$$t_1 = t\uparrow(\Lambda\cup\Lambda_1) \wedge t_2 = t\uparrow(\Lambda\cup\Lambda_2) \Leftrightarrow (\sigma_1, \sigma_2) \in \mathcal{A}/t$$

where $t\uparrow\Lambda$ denotes the trace t restricted to the events in Λ. This fact is simple to prove by induction on the lenght of the traces.

Lemma 2.13 Given acceptance automata $\mathcal{A}_1 = (\Lambda \cup \Lambda_1, \Sigma_1, \iota_1, M_1, \rightarrow_1)$ and $\mathcal{A}_2 = (\Lambda \cup \Lambda_2, \Sigma_2, \iota_2, M_2, \rightarrow_2)$ its parallel composition satisfies:

$$\mathcal{F}(\mathcal{A}_1 \parallel \mathcal{A}_2) = \mathcal{F}(\mathcal{A}_1) \parallel \mathcal{F}(\mathcal{A}_2)$$

Proof Let $(t, X) \in \mathcal{F}(\mathcal{A}_1) \parallel \mathcal{F}(\mathcal{A}_2)$, then from the definition of parallel composition in the failures model, there exist $(t_1, X_1) \in \mathcal{F}(\mathcal{A}_1)$ and $(t_2, X_2) \in \mathcal{F}(\mathcal{A}_2)$ such that:

$$t\uparrow\Lambda \cup \Lambda_1 = t_1, t\uparrow\Lambda \cup \Lambda_2 = t_2, X = X_1 \cup X_2$$

Now, from definitions 2.1 and 2.7, there are $\sigma_1 \in \mathcal{A}_1/t_1$ (resp. $\sigma_2 \in \mathcal{A}_2/t_2$) and menu $m_1 \in M_1(\sigma_1)$ (resp. $m_2 \in M_2(\sigma_2)$) such that:

$$X_1 \subseteq (\Lambda \cup \Lambda_1) - m_1 \text{ (resp. } X_2 \subseteq (\Lambda \cup \Lambda_2) - m_2)$$

Thus, from definition 2.10 and fact 2.15:

$$(\sigma_1, \sigma_2) \in (\mathcal{A}_1 \parallel \mathcal{A}_2)/t$$

and

$$m = (m_1 \cap m_2) \cup (\Lambda_1 \cap m_1) \cup (\Lambda_2 \cap m_2) \in M((\sigma_1, \sigma_2)).$$

It is easy to see that:

$$X = X_1 \cup X_2 \subseteq (\Lambda \cup \Lambda_1 \cup \Lambda_2) - m$$

so by definition 2.7, $(t, X) \in \mathcal{F}(\mathcal{A}_1 \parallel \mathcal{A}_2)$. The inverse holds for a similar reasoning. ◊

In what follows, we define the hiding operator on acceptance automata. As it was pointed above, divergent processes can not be modelled by means of acceptance automata. Since the hiding operator might cause divergence, even when applied to non-divergent processes, it is impossible to define a total operator over acceptance automata, while preserving the failures semantics. In order to overcome this difficulty, we first introduce the notion of divergence-free automaton, and then define hiding as a partial operator.

Definition 2.14 [Jos 88] Given an acceptance automaton $\mathcal{A} = (\Lambda, \Sigma, \iota, M, \rightarrow)$, and a set of events $\Lambda_h \subseteq \Lambda$, we say that $\mathcal{A}$ is *divergence free* with respect to Λ_h, if the following holds:

$$\forall\, s \in \mathrm{traces}(\mathcal{A}).\ \neg\forall\, n \in \mathbb{N}.\ \exists\, t \in \Lambda_h{}^*.\ \#t > n \wedge st \in \mathrm{traces}(\mathcal{A})$$

Lemma 2.15 The acceptance automaton $\mathcal{A} = (\Lambda \cup \Lambda_h, \Sigma, \iota, M, \rightarrow)$, is divergence free with respect to the set of events Λ_h iff there exists a well founded set $(\Omega, <)$ and a metric:

$$\omega\colon \mathrm{reach}(\mathcal{A}) \rightarrow \Omega$$

satisfying:

$$\forall\, \tau \in \Lambda_h,\ \sigma \rightarrow_\tau \sigma'.\ \omega(\sigma') < \omega(\sigma)$$

This is a simple consequence of the definition of well-founded set.

Definition 2.16 Given an acceptance automaton $\mathcal{A} = (\Lambda \cup \Lambda_h, \Sigma, \iota, M, \rightarrow)$, which is divergence free with respect to the set of events Λ_h, $\mathcal{A}$ *after hiding* Λ_h, denoted by $\mathcal{A}\backslash\Lambda_h$, $= (\Lambda, \Sigma, \iota, M_h, \rightarrow_h)$, is defined as follows:

1) $M_h \subseteq \Sigma \times \mathcal{P}(\Lambda)$, is the smallest relation satisfying:

   $\forall\, \sigma \in \Sigma, m \in M(\sigma).\ (\Lambda_h \cap m) = \emptyset \Rightarrow m \in M_h(\sigma)$

   $\forall\, \sigma_1, \sigma_2 \in \Sigma, m_1 \in M(\sigma_1), m_2 \in M_h(\sigma_2), \tau \in (\Lambda_h \cap m_1).$

   $\quad \sigma_1 \rightarrow_\tau \sigma_2 \Rightarrow (m_1 - \Lambda_h) \cup m_2,\ m_2 \in M_h(\sigma_1)$

2) $\rightarrow_h \subseteq \Sigma \times \Lambda \times \Sigma$, is the smallest relation satisfying:

   $\forall\, e \in \Lambda, \sigma_1 \rightarrow_e \sigma_2.\ \sigma_1 \rightarrow_{he} \sigma_2$

   $\forall\, e \in \Lambda, \tau \in \Lambda_h, \sigma_1 \rightarrow_\tau \sigma, \sigma \rightarrow_e \sigma_2.\ \sigma_1 \rightarrow_{he} \sigma_2$

Although this definition is recursive, $\mathcal{A}\backslash\Lambda_h$ is well-defined unless $\mathcal{A}$ is not divergence free with respect to the set of events Λ_h. Otherwise, if $\mathcal{A}$ is divergence free then there might exist paths of inestable states, the last element in such a path being a stable state. However, there will never exist closed paths made up of inestable states. Basically, this definition causes the copy of menus from the final state up to the initial state in such a path.

Fact 2.17 If $\mathcal{A}$ is an acceptance automaton, which is divergence free with respect to the set of events Λ_h,then $\mathcal{A}\backslash\Lambda_h$ is an acceptance automaton.

<u>Fact 2.18</u> If $\mathcal{A} = (\Lambda \cup \Lambda_h, \Sigma, \iota, M, \rightarrow)$ is divergence free with respect to the set of events Λ_h, then the following predicates hold:

$\forall t \in \text{traces}(\mathcal{A}).\ t\uparrow\Lambda \in \text{traces}(\mathcal{A}\backslash\Lambda_h)$

$\forall t \in \text{traces}(\mathcal{A}), \sigma \in \mathcal{A}/t, m \in M(\sigma).$

$$m \cap \Lambda_h = \emptyset \Rightarrow (\exists\, \sigma_h \in (\mathcal{A}\backslash\Lambda_h)/(t\uparrow\Lambda).\ m \in M_h(\sigma_h))$$

This is simple to prove by induction on the lenght of t.

<u>Lemma 2.19</u> Given an acceptance automaton $\mathcal{A} = (\Lambda \cup \Lambda_h, \Sigma, \iota, M, \rightarrow)$, which is divergence free with respect to the set of events Λ_h: $\mathfrak{F}(\mathcal{A}\backslash\Lambda_h) = \mathfrak{F}(\mathcal{A})\backslash\Lambda_h$

<u>Proof</u> Let $(t_h, X_h) \in \mathfrak{F}(\mathcal{A})\backslash\Lambda_h$, then, from the definition of the hiding operator in the failures model, there exists $(t, X) \in \mathfrak{F}(\mathcal{A})$, such that:

$t\uparrow\Lambda = t_h, X = X_h \cup \Lambda_h$

Now, from definitions 2.1 and 2.7, there are $\sigma \in \mathcal{A}/t$ and menu $m \in M(\sigma)$ such that:

$X = X_h \cup \Lambda_h \subseteq \Lambda \cup \Lambda_h - m$

Thus, $m \cap \Lambda_h = \emptyset$. Then from fact 2.18 there exists a state $\sigma_h \in (\mathcal{A}\backslash\Lambda_h)/t_h$, such that $m \in M_h(\sigma_h)$. So by definition 2.7, $(t_h, X_h) \in \mathfrak{F}(\mathcal{A}\backslash\Lambda_h)$. The inverse holds for a similar reasoning. ◊

<u>3. Standard Acceptance Automata</u>

In the previous section, the basic notions and operations concerning acceptance automata have been introduced. It is important to remark that acceptance automata, as defined in 2.1, might exhibit some non-desirable features. For instance, they can include non-reachable states. As it will be discussed below, this was necessary in order to define the most interesting operations: parallel composition and hiding. In this section we select a particular class of acceptance automata, the so-called standard acceptance automata, that will serve as the basis both for defining and comparing processes, possibly in a mechanical way. Through this section we assume an acceptance automaton $\mathcal{A} = (\Lambda, \Sigma, \iota, M, \rightarrow)$.

<u>Definition 3.1</u> $\mathcal{A}$ is said to be *junk free* if:

$\forall \sigma \in \Sigma.\ \exists t \in \text{traces}(\mathcal{A}).\ \sigma \in \mathcal{A}/t$

<u>Fact 3.2</u> We can always define the corresponding junk free acceptance automaton junk-free($\mathcal{A}$) = $(\Lambda, \text{reach}(\mathcal{A}), \iota, M^*, \rightarrow^*)$, with the same failures semantics as $\mathcal{A}$. M^* is defined as the smallest relation satisfying:

$\forall \sigma \in \text{reach}(\mathcal{A}), m \in M(\sigma).\ m \in M^*(\sigma)$

Similarly, $\rightarrow^*$ is defined as the smallest relation satisfying

$\forall \sigma, \sigma' \in \text{reach}(\mathcal{A}), e \in \Lambda.\ \sigma \rightarrow_e \sigma' \Rightarrow \sigma \rightarrow^*_e \sigma'$

Obviously, since there is no interest in considering non-reachable states, we could try to restrict the definition of acceptance automata to those tuples with this property. Nevertheless, it must be observed that the parallel composition of junk-free automata does not define, in general, another junk-free automata. The same holds for the hiding operator. Thus, if we want to define those operations over acceptance automata, it must be posible for them to have non-reachable states. This is of no concern because, as it has been pointed above, it is always possible to define the equivalent junk-free automata.

Definition 3.3 $\mathcal{A}$ is said to be *unambiguous* if:

$\forall t \in \text{traces}(\mathcal{A}).\ \left|\mathcal{A}/t\right| = 1$

We will use the term *ambiguous* when referring to an acceptance automaton that is not unambiguous.

The following examples define an ambiguous and an unambiguous automaton, both with the same failures semantics:

$\Lambda = \{a, b\}$
$\Sigma = \{0, 1, 2\}$
$\iota = 0$
$M = \{(0, \{a\}), (1, \{b\}), (2, \emptyset)\}$
$\rightarrow = \{(0, a, 1), (0, a, 2), (1, b, 2)\}$

$\Lambda = \{a, b\}$
$\Sigma = \{0, 1, 2\}$
$\iota = 0$
$M = \{(0, \{a\}), (1, \emptyset), (1, \{b\}), (2, \emptyset)\}$
$\rightarrow = \{(0, a, 1), (1, b, 2)$

As it is shown in this example, the process defined by an unambiguous acceptance automaton is not in general deterministic. Moreover, the process defined by an ambiguous acceptance automaton could be deterministic. This motivates our next definition:

Definition 3.4 $\mathcal{A}$ is said to be a *deterministic automaton* if it is unambiguous and:

$\forall t \in \text{traces}(\mathcal{A}), \sigma \in \mathcal{A}/t.\ \left|M(\sigma)\right| = 1$

Fact 3.5 The parallel composition of unambiguous automata defines another unambiguous automata. This is trivial from the definition of the parallel composition, and in particular, from the definition of its transition relation.

Observe that the hiding operation applied to an unambiguous acceptance automaton does not define, in general, another unambiguous acceptance automaton. The menu relation was included in the definition of acceptance automata to model the non-deterministic behaviour of concurrent systems. The intuitive idea is that non-determinism should be represented by the existence of several menus in a given state. However, the use of ambiguous acceptance automata allows the modelling of non-deterministic behaviour, even when there is a unique menu associated with every reachable state. According to this, it might seem appropiate to restrict the definition of acceptance automata to those which are unambiguous. This is not possible if we want to define the hiding operator over acceptance automata. In what follows we show how to deal with this situation defining the equivalent unambiguous acceptance automaton.

Definition 3.6 The *unambiguous form* of $\mathcal{A}$ is the acceptance automaton $\mathcal{A}_{uf} = (\Lambda, \Sigma_{uf}, \iota_{uf}, M_{uf}, \rightarrow_{uf})$ defined as follows:

- $\Sigma_{uf} = traces(\mathcal{A}) / \approx$,
  where $\approx \subseteq traces(\mathcal{A}) \times traces(\mathcal{A})$, is defined as the smallest relation satisfying:
  $\forall\, t_1, t_2 \in traces(\mathcal{A}).\ \mathcal{A}/t_1 = \mathcal{A}/t_2 \Rightarrow t_1 \approx t_2$
- $\iota_{uf} = [\langle\rangle]$
- $M_{uf} \subseteq \Sigma_{uf} \times \mathcal{P}(\Lambda)$, is the smallest relation satisfying:
  $\forall\, t \in traces(\mathcal{A}), \sigma \in \mathcal{A}/t, m \in M(\sigma).\ m \in M_{uf}([t])$
- $\rightarrow_{uf} \subseteq \Sigma_{uf} \times \Lambda \times \Sigma_{uf}$, is the smallest relation satisfying:
  $\forall\, t \in traces(\mathcal{A}), e \in \Lambda.\ te \in traces(\mathcal{A}) \Rightarrow [t] \rightarrow_{uf_e} [te]$

Lemma 3.7 Given an ambiguous acceptance automaton $\mathcal{A}$ and its unambiguous form $\mathcal{A}_{df}$: $\mathcal{A} \approx \mathcal{A}_{df}$

For finite acceptance automata, there is an effective procedure to build its unambiguous form.

Definition 3.8 $\mathcal{A}$ is said to be *saturated* if:

$\forall\, \sigma \in \Sigma, m \in \mathcal{P}(\Lambda), m' \in M(\sigma).\ m' \subseteq m \subseteq next(\sigma) \Rightarrow m \in M(\sigma)$

Note that the parallel composition and hiding of saturated automata define another saturated automata. Moreover, given a non-saturated acceptance automaton, the derivation of its saturated form is trivial and it is easy to prove that its failures semantics is the same. This fact is directly reflected in the TCSP laws.

Definition 3.9 Given an acceptance automaton $\mathcal{A}$, we say that it is in *standard form* if it is junk-free, unambiguous and saturated.

This is the most important concept in this section. It must be observed that for finite state acceptance automata we can build mechanically its standard form. Moreover, the result of this transformation process is unique, regardless of the definition of the original automaton. The main interest of this reduction procedure will be made clear in the next section.

4. Minimal Automaton

Given a non divergent TCSP process $\mathbb{P}$, there are many standard automata exhibiting its behaviour. The purpose of this section is to explore the category SAut ($\mathbb{P}$) of all the standard automata that have $\mathbb{P}$ as semantics. The morphisms will be a restricted version of the classical notion of bisimulation that we call a *simulation*. We show that in this category there exist both initial and final objects. The initial one or *maximal automaton* is the automaton with the greatest number of states corresponding to the TCSP normal form [Nic 85]. The final one or *minimal automaton* is the automaton with the least number of states. For practical purposes this is the most interesting one. For finite acceptance automata there is an efective procedure to compute their final versions.

Definition 4.1 Given a non-divergent process in the Failures Model $\mathbb{P} = (\Lambda, \mathfrak{F}, \mathfrak{D})$, with $\mathfrak{F} \subseteq \Lambda^* \times \mathcal{P}(\Lambda)$ and $\mathfrak{D} = \emptyset$, its *maximal acceptance automaton* $\mathfrak{A}_I(\mathbb{P}) = (\Lambda, \Sigma, \iota, M, \rightarrow)$, is defined as follows:

- Λ is the alphabet of $\mathbb{P}$,
- $\Sigma = \{t \in \Lambda^*, (t, \emptyset) \in \mathbb{P}\}$,
- ι is the empty trace $\langle\rangle$
- $M \subseteq \Sigma \times \mathcal{P}(\Lambda)$, is the smallest relation satisfying:
  $\forall (\sigma, F) \in \mathbb{P}. (\forall (\sigma, F') \in \mathbb{P}. F \not\subset F') \Rightarrow \neg F \in M(\sigma)$
  $\forall \sigma \in \Sigma, m \in \mathcal{P}(\Lambda), m' \in M(\sigma). m' \subseteq m \subseteq \text{next}(\sigma) \Rightarrow m \in M(\sigma)$
- $\rightarrow \subseteq \Sigma \times \Lambda \times \Sigma$, is the smallest relation satisfying:
  $\forall (\sigma, \emptyset), (\sigma e, \emptyset) \in \mathbb{P}. \sigma \rightarrow_e \sigma e$

The first line in the menu relation introduces as menus the complements of the maximal refusal sets after a given trace. The second line is the convex closure of the previous menus [HoJi 85]. It is interesting to note that the menu relation conveys more information than an alternative *refusal relation* as used for instance in [AGS 90]. From the first one it is immediate to obtain the second one (see definition 2.7). To do the opposite we need next(σ), the set of possible events after a trace σ, i.e. we need also to look at transition relation, i.e. at the whole failures set.

It is obvious to see that this definition satisfies the constraints of an acceptance automaton, in particular those of a standard one, and that its failures semantics coincides exactly with the original process $\mathbb{P}$. In fact, this maximal automaton is exactly the TCSP term in normal form that can be constructed from the failures set as in [Nic 85].

Now we proceed to the construction of the category: given the non-divergent TCSP process $\mathbb{P}$, let SAut ($\mathbb{P}$) be the class of all the standard acceptance automata that have $\mathbb{P}$ as semantics.

Definition 4.2 Given $\mathcal{A}_1 = (\Lambda, \Sigma_1, \iota_1, M_1, \rightarrow_1)$, $\mathcal{A}_2 = (\Lambda, \Sigma_2, \iota_2, M_2, \rightarrow_2)$ standard automata, we say that a mapping $f: \Sigma_1 \rightarrow \Sigma_2$ is a *morphism* or a *simulation* from $\mathcal{A}_1$ to $\mathcal{A}_2$ if it satisfies the following conditions:

1) $f(\iota_1) = \iota_2$
2) $\forall \sigma \in \Sigma_1.\ M_1(\sigma) = M_2(f(\sigma))$
3) $\forall \sigma \rightarrow_{1_e} \sigma'.\ f(\sigma) \rightarrow_{2_e} f(\sigma')$

A simulation is a particular case of the classical notion of bisimulation that can be adapted to acceptance automata in the following way:

Definition 4.3 Given standard acceptance automata $\mathcal{A}_1 = (\Lambda, \Sigma_1, \iota_1, M_1, \rightarrow_1)$ and $\mathcal{A}_2 = (\Lambda, \Sigma_2, \iota_2, M_2, \rightarrow_2)$ a *bisimulation of* $\mathcal{A}_1$ *and* $\mathcal{A}_2$ is a relation $\mathfrak{B} \subseteq \Sigma_1 \times \Sigma_2$, satisfying:

1) $(\iota_1, \iota_2) \in \mathfrak{B}$
2) $\forall (\sigma_1, \sigma_2) \in \mathfrak{B}.\ M_1(\sigma_1) = M_2(\sigma_2)$
3) $\forall (\sigma_1, \sigma_2) \in \mathfrak{B}, e \in \Lambda, \sigma_1 \rightarrow_{1_e} \sigma, \sigma_2 \rightarrow_{2_e} \sigma'.\ (\sigma, \sigma') \in \mathfrak{B}$

The difference between a general relation and a mapping is that the last one will allow us to go from automata with more states to automata with less states.

Lemma 4.4 Given standard acceptance automata $\mathcal{A}_1$ and $\mathcal{A}_2$, $\mathcal{A}_1 \approx \mathcal{A}_2$ iff there exists a bisimulation between $\mathcal{A}_1$ and $\mathcal{A}_2$.

Proof Let us first see the following fact :
If $\mathfrak{B}$ is a bisimulation of $\mathcal{A}_1$ and $\mathcal{A}_2$ then the following predicate holds:

$\forall t \in \text{traces}(\mathcal{A}_1).\ (\mathcal{A}_1/t, \mathcal{A}_2/t) \in \mathfrak{B}$

Where $\mathcal{A}_1/t$, $\mathcal{A}_2/t$ denote the unique state reached after the trace t. This is simple to prove by induction on the lenght of t. From this fact and definition 2.7, it follows that if there exists a bisimulation of $\mathcal{A}_1$ and $\mathcal{A}_2$ then $\mathcal{A}_1 \approx \mathcal{A}_2$. The opposite is also simple to prove. Given that $\mathcal{A}_1 \approx \mathcal{A}_2$, let us define $\mathfrak{B} \subseteq \Sigma_1 \times \Sigma_2$ to be the smallest relation such that:

$\forall t \in \text{traces}(\mathcal{A}_1), \sigma_1 \in \mathcal{A}_1/t, \sigma_2 \in \mathcal{A}_2/t \Rightarrow (\sigma_1, \sigma_2) \in \mathfrak{B}$

$\mathfrak{B}$ trivially satisfies conditions 1 and 3 of definition 4.3. Then from $\mathcal{A}_1 \approx \mathcal{A}_2$ and the fact that they are standard, it comes condition 2. So $\mathfrak{B}$ is a bisimulation. ◊

It its easy to prove that the composition of morphisms gives another morphism and that it is associative with the identity morphism being the identity mapping. So SAut ($\mathbb{P}$) together with all the morphisms defined between them turns out to be a category.

In SAut ($\mathbb{P}$), an isomorphism will be a bijective morphism and two isomorphic automata will be in fact equal up to renaming of states. As usual, there will be a morphism from the initial object (if it exists) to any other object in the category and from any object to the final one (if it exists). The initial and final objects (if they exist) are unique up to isomorphism. We shall see that these objects exist in SAut ($\mathbb{P}$).

<u>Lemma 4.5</u> Given $\mathbb{P}$, the maximal acceptance automaton $\mathcal{A}_I$ ($\mathbb{P}$) is initial in SAut ($\mathbb{P}$).

<u>Proof:</u> Given any acceptance automaton $\mathcal{A} \in$ SAut ($\mathbb{P}$) the mapping f:traces($\mathcal{A}_I$ ($\mathbb{P}$)) $\rightarrow \Sigma$, defined by f(t) = $\mathcal{A}$/t, where $\mathcal{A}$/t denotes the unique state reached after the trace t, is a morphism and it is unique. This follows from lemma 4. 4. ◊

Now we proceed to the proof of existence and construction of the final automaton $\mathcal{A}_F$ ($\mathbb{P}$) of SAut($\mathbb{P}$). First, we need the concept of *congruence* and of *quotient automaton* by a congruence.

<u>Definition 4.6</u> Given an standard acceptance automaton $\mathcal{A}$ = (Λ, Σ, ι, M, $\rightarrow$), a *congruence* in $\mathcal{A}$ is an equivalence relation $\equiv \subseteq \Sigma \times \Sigma$ satisfying the following properties:

1) $\forall\, \sigma_1, \sigma_2 \in \Sigma.\ \sigma_1 \equiv \sigma_2 \Rightarrow M(\sigma_1) = M(\sigma_2)$
2) $\forall\, \sigma_1, \sigma_2 \in \Sigma, e \in \Lambda. \sigma_1 \equiv \sigma_2 \wedge \sigma_1 \rightarrow_e \sigma \wedge \sigma_2 \rightarrow_e \mu \Rightarrow \sigma \equiv \mu$

<u>Definition 4.7</u> Given an standard acceptance automaton $\mathcal{A}$ = (Λ, Σ, ι, M, $\rightarrow$) and a congruence $\equiv$ in $\mathcal{A}$ the *quotient acceptance automaton* of $\mathcal{A}$ by $\equiv$, denoted by $\mathcal{A}/\!\equiv$ = (Λ, $\Sigma_\equiv$, $\iota_\equiv$, $M_\equiv$, $\rightarrow_\equiv$) is defined as follows:

1) $\Sigma_\equiv = \Sigma/Q$, is the quotient set of Σ by $\equiv$
2) $\iota_\equiv = [\iota]$
3) $M_\equiv([\sigma]) = M(\sigma)$
4) $[\sigma] \rightarrow_e [\sigma']$ iff $\sigma \rightarrow_e \sigma'$

It is inmediate to show that this definition is independent of the representative σ chosen for the class [σ]. There is a strong connection between congruences and simulations as the following lemmas show:

<u>Lemma 4.8</u> Given an acceptance automata $\mathcal{A}$ = (Λ, Σ, ι, M, $\rightarrow$) and a congruence $\equiv$ in $\mathcal{A}$, the mapping f: $\mathcal{A} \rightarrow \mathcal{A}/\!\equiv$ defined by f(σ) = [σ] is a simulation.

<u>Proof</u> Properties 1, 2, and 3 of definition 4.2 are a direct translation of properties 2, 3, and 4 of definition 4.7. ◊

<u>**Lemma 4.9**</u> Given acceptance automata $\mathcal{A}_1 = (\Lambda, \Sigma_1, \iota_1, M_1, \rightarrow_1)$ and $\mathcal{A}_2 = (\Lambda, \Sigma_2, \iota_2, M_2, \rightarrow_2)$, with $\mathcal{A}_1, \mathcal{A}_2 \in$ SAut ($\mathbb{P}$), if there is a simulation f: $\mathcal{A}_1 \rightarrow \mathcal{A}_2$, then the equivalence relation in Σ_1 defined by:

$\sigma_1 \equiv \sigma_2$ iff $f(\sigma_1) = f(\sigma_2)$

is a congruence in $\mathcal{A}_1$ and $\mathcal{A}_1/\equiv$ is isomorphic to $\mathcal{A}_2$.

<u>Proof</u>

1) $\equiv$ is a congruence:
condition 1 of definition 4.6 follows directly from condition 2 of definition 4.2. With respect to condition 2 of definition 4.6, let us assume:

$\sigma_1 \rightarrow_{1e} \sigma_1', \sigma_2 \rightarrow_{2e} \sigma_2'$ and $f(\sigma_1) = f(\sigma_2)$

by being f a simulation we have

$f(\sigma_1) \rightarrow_{2e} f(\sigma_2')$ and $f(\sigma_2) \rightarrow_{2e} f(\sigma_2')$

then, as $\mathcal{A}_2$ is standard $f(\sigma_1') = f(\sigma_2')$ so $\sigma_1' \equiv \sigma'$

2) First observe that:

$\forall\, \sigma_2 \in \Sigma_2.\ \exists\, t \in$ traces($\mathcal{A}_2$). $\{\sigma_2\} = \mathcal{A}_2/t$

and as f is a morphism: $f(\sigma_1) = \sigma_2$, where $\{\sigma_1\} = \mathcal{A}_1/t$. So f is surjective.

Then, the mapping f*: $\mathcal{A}_1/\equiv \rightarrow \mathcal{A}_2$ defined by:

$f^*([\sigma]) = f(\sigma)$

is a biyective morphism and $\mathcal{A}_1/\equiv$ is isomorphic to $\mathcal{A}_2$. ◊

As a consequence, doing the quotient of an automaton by a congruence, preserves the behaviour.

<u>**Definition 4.10**</u> Given an acceptance automata $\mathcal{A} = (\Lambda, \Sigma, \iota, M, \rightarrow)$ and two congruences $\equiv_1, \equiv_2$ in $\mathcal{A}$, the *sum* $\equiv_1 + \equiv_2 \subseteq \Sigma \times \Sigma$ is defined as the transitive closure of $\equiv_1 \cup \equiv_2$.

<u>**Fact 4.11**</u> Given an acceptance automata $\mathcal{A} = (\Lambda, \Sigma, \iota, M, \rightarrow)$ and two congruences $\equiv_1, \equiv_2$ in $\mathcal{A}$, the *sum* $\equiv_1 + \equiv_2$ is another congruence on $\mathcal{A}$.

Since the identity mapping on states is a congruence, and since the sum and intersection of two congruences is another congruence, the set (Cong($\mathcal{A}$), $\subseteq$), with Cong($\mathcal{A}$) denoting the family of congruences in $\mathcal{A}$, turns out to be a complete lattice. Let us now denote by $\equiv_F(\mathcal{A})$ the maximum congruence on $\mathcal{A}$ constructed by suming up all the congruences on $\mathcal{A}$.

<u>**Definition 4.12**</u> Given a non-divergent process TCSP process $\mathbb{P}$ its *minimal acceptance automaton* $\mathcal{A}_F(\mathbb{P})$ is defined by $\mathcal{A}_I(\mathbb{P})/\equiv_F(\mathcal{A}_I(\mathbb{P}))$. Given an acceptance automaton $\mathcal{A}$, we will use the term *normal form*, denoted by $\mathcal{A}\downarrow$, when referring to the corresponding minimal

automaton.

Lemma 4.13 $\mathcal{A}_F(\mathbb{P})$ is final in SAut($\mathbb{P}$).

Proof We need to show that for all $\mathcal{A}$ in SAut($\mathbb{P}$), there is a unique morphism f: $\mathcal{A} \to \mathcal{A}_F(\mathbb{P})$. As $\mathcal{A}_I(\mathbb{P})$ is initial, there exists a unique morphism f_I: $\mathcal{A}_I(\mathbb{P}) \to \mathcal{A}$ and congruence $\equiv_I$ such that $\mathcal{A}_I(\mathbb{P})/\equiv_I$ is isomorphic to $\mathcal{A}$. Also there is a unique morphism f_F: $\mathcal{A}_I(\mathbb{P}) \to \mathcal{A}_F(\mathbb{P})$ induced by $\equiv_F(\mathcal{A}_I(\mathbb{P}))$. As this is the maximum congruence in Cong($\mathcal{A}_I(\mathbb{P})$), we have that $\equiv_I \subseteq \equiv_F(\mathcal{A}_I(\mathbb{P}))$ and the mapping f: $\mathcal{A} \to \mathcal{A}_F(\mathbb{P})$, defined by $f([t]_I) = [t]_F$ for any trace t, is well-defined. It is straightforward to show that f is a morphism, and since $f_F = f \circ f_I$, f must also be unique. ◊

Fact 4.14 If $\mathcal{A}(\mathbb{P})$ is finite, then the procedure for computing $\mathcal{A}_F(\mathbb{P})$ is algorithmic. The procedure will be an adapted version of the Moore algorithm [Woo 87].

This fact provides in many situations an effective mean to verify the correctness of parallel systems. The complete method, already advanced in [PA 89], will consist of the following steps:

- specify the system at the outermost level. Construct the acceptance automaton $\mathcal{A}_{sp}$ of the specification. Let us assume that it is finite
- implement the system as a parallel composition of automata $\mathcal{A}_1, .., \mathcal{A}_n$, with synchronization alphabet Λ
- use the definitions in this paper to compute the automaton $\mathcal{A}_{imp} = (\mathcal{A}_1 \parallel .. \parallel \mathcal{A}_n)\backslash\Lambda$
- verify that the normal forms of $\mathcal{A}_{sp}$ and $\mathcal{A}_{imp}$ are isomorphic, $\mathcal{A}_{sp}\!\downarrow = \mathcal{A}_{imp}\!\downarrow$

5. Refinements

Usually the implementation of a system is obtained by first composing in parallel a family of subsystems and then hiding those events corresponding to their internal activity. Asking the implementation to behave exactly like the specification amounts to say that their initial (respectively final) forms are isomorphic. Nevertheless this is a too strong requirement in most situations. For this reason we will adopt the notion of refinement as defined in the Failures Model, and provide an alternative characterization in terms of acceptance automata that will be amenable to mechanical verification.

Definition 5.1 Given acceptance automata $\mathcal{A}_{sp}$ and $\mathcal{A}_{imp}$, $\mathcal{A}_{imp}$ is said to be a *refinement* of $\mathcal{A}_{sp}$, denoted by $\mathcal{A}_{imp} \sqsubseteq \mathcal{A}_{sp}$, if $\mathcal{F}(\mathcal{A}_{imp}) \subseteq \mathcal{F}(\mathcal{A}_{sp})$.

Definition 5.2 Given an acceptance automaton $\mathcal{A} = (\Lambda, \Sigma, \iota, M, \to)$ it is said to be *dead-lock free* if: $\forall\, \sigma \in$ reach($\mathcal{A}$). $\emptyset \notin M(\sigma)$

Lemma 5.3 Let $\mathcal{A}_{sp} = (\Lambda, \Sigma_{sp}, \iota_{sp}, M_{sp}, \rightarrow_{sp})$ and $\mathcal{A}_{imp} = (\Lambda, \Sigma_{imp}, \iota_{imp}, M_{imp}, \rightarrow_{imp})$ be initial acceptance automata:

$$\mathcal{A}_{imp} \sqsubseteq \mathcal{A}_{sp} \text{ iff } \forall\, t \in \mathrm{traces}(\mathcal{A}_{imp}).\ (t \in \mathrm{traces}(\mathcal{A}_{sp}) \wedge M_{imp}(t) \subseteq M_{sp}(t))$$

where $M_{imp}(t)$, $M_{sp}(t) \subseteq \mathcal{P}(\Lambda)$, denote families of menus.

A similar fact might be stated by comparing the respective final forms. In this case, a means for relating states in the specification and implementation must be provided in the form of a relation over states. Most often, this relation will be a mapping even though this is not the general case.

Fact 5.4 Let $\mathcal{A}_{sp} = (\Lambda, \Sigma_{sp}, \iota_{sp}, M_{sp}, \rightarrow_{sp})$ and $\mathcal{A}_{imp} = (\Lambda, \Sigma_{imp}, \iota_{imp}, M_{imp}, \rightarrow_{imp})$ be final acceptance automata. $\mathcal{A}_{imp}$ is a refinement of $\mathcal{A}_{sp}$ if there exists a relation $\Phi \subseteq \Sigma_{imp} \times \Sigma_{sp}$ satisfying:

- $(\iota_{imp}, \iota_{sp}) \in \Phi$,
- $\forall\, (\sigma_{imp}, \sigma_{sp}) \in \Phi.\ M_{imp}(\sigma_{imp}) \subseteq M_{sp}(\sigma_{sp})$
- $\forall\, (\sigma_{imp}, \sigma_{sp}) \in \Phi, \sigma_{imp} \rightarrow_{imp_e} \mu_{imp}, \sigma_{sp} \rightarrow_{sp_e} \mu_{sp}.\ (\mu_{imp}, \mu_{sp}) \in \Phi$

We shall use the term *abstraction relation* when referring to this relation.

Fact 5.5 Given acceptance automata $\mathcal{A}_{sp}$ and $\mathcal{A}_{imp}$, $\mathcal{A}_{imp}$ is dead-lock free iff $\mathcal{A}_{sp}$ is dead-lock free and $\mathcal{A}_{imp}$ is a refinement of $\mathcal{A}_{sp}$.

This is a general fact concerning refinements and coming from the Failures Model. Besides that, fact 5.4 provides an effective procedure for proving correctness of refinements when dealing with finite state systems. Although this is the case with a great number of interesting problems, finite state systems are far from being the general situation. The following fact attempts to handle this problem.

Fact 5.6 Let $\mathcal{A}_{sp} = (\Lambda, \Sigma_{sp}, \iota_{sp}, M_{sp}, \rightarrow_{sp})$ and $\mathcal{A}_{imp} = (\Lambda \cup \Lambda_h, \Sigma_{imp}, \iota_{imp}, M_{imp}, \rightarrow_{imp})$ be standard acceptance automata. Provided that $\mathcal{A}_{imp}$ is divergence free w.r.t Λ_h, then $\mathcal{A}_{imp}\backslash\Lambda_h$ is a refinement of $\mathcal{A}_{sp}$ iff there is a mapping $\Phi \subseteq \Sigma_{imp} \times \Sigma_{sp}$ satisfying:

- $(\iota_{imp}, \iota_{sp}) \in \Phi$,
- $\forall\, (\sigma_{imp}, \sigma_{sp}) \in \Phi, m \in M_{imp}(\sigma_{imp}).$
  $m \cap \Lambda_h = \emptyset \Rightarrow m \in M_{sp}(\sigma_{sp})$
  $m \cap \Lambda_h \neq \emptyset \Rightarrow m - \Lambda_h\ \mathrm{next}(\sigma_{sp})$
- $\forall\, (\sigma_{imp}, \sigma_{sp}) \in \Phi, \sigma_{imp} \rightarrow_{imp_e} \mu_{imp}, \sigma_{sp} \rightarrow_{sp_e} \mu_{sp}.\ (\mu_{imp}, \mu_{sp}) \in \Phi$
- $\forall\, (\sigma_{imp}, \sigma_{sp}) \in \Phi, \tau \in \Lambda_h, \sigma_{imp} \rightarrow_{imp_\tau} \mu_{imp}.\ (\mu_{imp}, \sigma_{sp}) \in \Phi$

Observe that if $\mathcal{A}_{sp}$ and $\mathcal{A}_{imp}$ are in initial form, then this relation is in fact a mapping $\mathbb{I}$: $\mathrm{traces}(\mathcal{A}_{imp}) \rightarrow \mathrm{traces}(\mathcal{A}_{sp})$, defined by $\mathbb{I}(t) = t\!\uparrow\!\Lambda$.

References

[AGS 90] Autebert, J.M., Gabarro, J., Serna, M.J. "Finite memory CSP and CCS devices." Internal Rport, Departamento de Lenguajes y Sistemas Informáticos, Universidad Politécnica de Cataluña (1990).

[BHR 84] Brookes, S.D., Hoare, C.A.R., Roscoe, A.W. "A Theory of Communicating Sequential Processes." *Journal of the ACM* **31** (1984) 560-599.

[Hen 85] Hennessy, M. "Acceptance Trees." *Journal of the ACM* **32** (1985) 896-928.

[Hen 88] Hennessy, M. *Algebraic Theory of Processes*. MIT Press (1988).

[Hoa 85] Hoare, C.A.R. *Communicating Sequential Processes*. Prentice-Hall (1985).

[HoJi 85] Hoare, C.A.R., Jifeng, H. "Algebraic Specification and Proof of Properties of Communicating Sequential Processes." *Tech. Rep.* PRG-**52** (1985) Oxford University Computing Laboratory.

[Jos 88] Josephs, M.B. "A state-based approach to communicating processes." *Distributed Computing* **3** (1988) 9 - 18.

[Nic 85] de Niccola, R. "Two Complete Axiom Systems for a Theory of Communicating Sequential Processes." *Information and Control* **64** (1985) 136-172.

[Par 81] Park, D. "Concurrency and automata on infinite sequences." *Proc. 5th GI Conf. of Theoretical Computer Science*, LNCS 104 (1981) 245-251.

[PA 89] Peña, R., Alonso, L.M. "Specification and Verification of TCSP Systems by Means of Partial Abstract Data Types." In *Proceedings* TAPSOFT'89, LNCS 352 (1989) 328-344.

[Woo 87] Wood, D. *Theory of computation*. John-Wiley (1987).