

A THEORY OF MAY TESTING FOR ACTORS

Prasanna Thati

Reza Ziaei

Gul Agha

University of Illinois at Urbana-Champaign, IL, USA

{thati,ziaei,agha}@cs.uiuc.edu

Abstract The Actor model and π -calculus have served as the basis of a large body of research on concurrency. We represent the Actor model as a typed asynchronous π -calculus, called $A\pi$. The type system imposes a certain discipline on the use of names to capture actor properties such as uniqueness and persistence. We investigate the notion of *may* testing in $A\pi$ and give a trace based characterization of it. Such a characterization simplifies reasoning about actor configurations as it does not involve quantification over all environments. We compare our characterization with that of asynchronous π -calculus, and highlight the differences that arise due to actor properties.

Keywords: Actors, interaction paths, may testing, π -calculus

1. Introduction

We introduce a basic calculus for the Actor model [1], called $A\pi$, and present a trace based characterization of may testing [7] in it. It is well-known that a trace based semantic characterization simplifies reasoning about equivalences, as it does not involve quantification over observing contexts. Both testing theories [2] and trace based models [16] have been studied for actors but the relation between them has not been investigated.

$A\pi$ is a typed asynchronous π -calculus [3, 8, 13], where the type system enforces properties specific to the Actor model. Since the operational semantics of π -calculus is unchanged, $A\pi$ can be seen as an embedding of the Actor model in π -calculus. This embedding not only provides a direct basis for comparison between the two models, but also enables us to apply concepts and techniques developed for π -calculus to Actors. Many formalisms for the Actor model have been proposed in the past [2, 6, 9, 15, 16] and various notions of equivalence have been

considered for them [2, 6, 16]. However, none of these formalisms is directly comparable to π -calculus. On the other hand, we believe reusing a well-known formalism provides some advantages over adopting a fresh approach.

We define a labeled transition system for $A\pi$ that reflects what is observable to an environment that interacts with an actor configuration. We use this transition system to derive an alternate characterization of may-testing in terms of the set of traces that an actor configuration can exhibit. The approach we adopt for establishing the characterization is similar to that used for asynchronous π -calculus [11]. The two characterizations differ in several respects due to differences in the notion of observability in actors and π -calculus, which are reflected in the labeled transition systems for the two calculi.

Due to space limitation we do not present the proofs, for which the reader is referred to [18]. We have also considered variants of $A\pi$ that differ in the name matching capabilities in [18], and sketched the key ideas behind trace based characterizations of may testing for them. These characterizations involve radical changes to the one presented in this paper.

2. The Actor Model

A computational system in the Actor Model, called a *configuration*, consists of a collection of concurrently executing actors and a collection of messages in transit [1]. Each actor has a unique name (the *uniqueness* property) and a behavior, and communicates with other actors via asynchronous messages. Actors are reactive in nature, i.e. they execute only in response to messages received. An actor's behavior is deterministic in that its response to a message is uniquely determined by the message contents. Message delivery in the Actor model is fair [4]. The delivery of a message can only be delayed for a finite but unbounded amount of time.

An actor can perform three basic actions on receiving a message: (a) create a finite number of actors with universally fresh names, (b) send a finite number of messages, and (c) assume a new behavior. Furthermore, all actions performed on receiving a message are concurrent; there is no ordering between any two of them. The following observations are in order here. First, actors are persistent in that they do not disappear after processing a message (the *persistence* property). Second, actors cannot be created with well known names or names received in a message (the *freshness* property).

3. The Calculus $\mathbf{A}\pi$

We assume an infinite set of names \mathcal{N} , and a set \mathcal{B} of behavior identifiers. We let u, v, w, x, y, z, \dots range over \mathcal{N} , and B range over \mathcal{B} . We write \tilde{x} for a tuple of names, and $len(\tilde{x})$ for the length of the tuple. For \tilde{x} of length n , x_i for $i \leq n$ denotes the i^{th} component of the tuple. We let C range over the set of preterms \mathcal{C} , which is defined by the following context-free grammar.

$$C := 0 \mid x(y).C \mid \bar{x}y \mid [x = y](C_1, C_2) \mid (\nu x)C \mid C_1|C_2 \mid B\langle\tilde{x};\tilde{y}\rangle$$

The order of precedence of combinators is the order in which they appear. The nil term 0 , represents an empty configuration. The output term $\bar{x}y$, represents a configuration with a single message targeted to x and with contents y . We call x the subject of the output term. The input term $x(y).C$ represents a configuration with an actor x whose behavior is $(y)C$. We call x the subject of the input term. The composition $C_1|C_2$ is a configuration containing all the actors and messages in C_1 and C_2 . The conditional $[x = y](C_1, C_2)$ is C_1 if x and y are the same names, and C_2 otherwise. The restriction $(\nu x)C$ is the same as C , except that x is now private to C . The term $B\langle\tilde{u};\tilde{v}\rangle$ is a behavior instantiation. The identifier B has a single defining equation of the form $B \stackrel{def}{=} (\tilde{x};\tilde{y})x_1(z).C$, where \tilde{x} is a tuple of distinct names of length 1 or 2, and \tilde{x}, \tilde{y} together contain exactly the free names in $x_1(z).C$. The definition provides a template for an actor behavior. For an instantiation $B\langle\tilde{u};\tilde{v}\rangle$ we assume $len(\tilde{u}) = len(\tilde{x})$, and $len(\tilde{v}) = len(\tilde{y})$.

Notational Conventions and Definitions. For a tuple \tilde{x} , we denote the set of names occurring in \tilde{x} by $\{\tilde{x}\}$. We write \tilde{x}, \tilde{y} for the result of appending \tilde{y} to \tilde{x} . We let \hat{z} range over $\{\emptyset, \{z\}\}$. By \tilde{x}, \hat{z} we mean \tilde{x}, z if $\hat{z} = \{z\}$, and \tilde{x} otherwise. The term $(\hat{z})C$ is $(\nu z)C$ if $\hat{z} = \{z\}$, and C otherwise. We write $(\nu x_1, \dots, x_n)C$ instead of $(\nu x_1)\dots(\nu x_n)C$.

The functions $fn(\cdot)$, $bn(\cdot)$, $n(\cdot)$ are defined on preterms the obvious way. Alpha equivalence on preterms, \equiv_α , is defined as usual. We also use the usual definition and notational convention for name substitution, and let σ range over substitutions. For a name x we write $\sigma(x)$ for the name to which x is mapped to by σ , and for a set of names S , we write $\sigma(S)$ to denote the set obtained by applying σ to each element of S . Name substitutions on configurations are defined modulo alpha equivalence, with the usual renaming convention to avoid captures. We write $C\sigma$ to denote the result of applying the simultaneous substitution σ to C .

Let $X \subset \mathcal{N}$. We assume $\perp, * \notin \mathcal{N}$, and define $X^* = X \cup \{\perp, *\}$. For $f : X \rightarrow X^*$, we define $f^* : X^* \rightarrow X^*$ as $f^*(x) = f(x)$ for $x \in X$ and $f^*(\perp) = f^*(*) = \perp$. Further, if σ is a substitution which is one-to-one on X , we define $f\sigma : \sigma(X) \rightarrow \sigma(X)^*$ as $f\sigma(\sigma(x)) = \sigma(f(x))$, where we let $\sigma(\perp) = \perp$ and $\sigma(*) = *$.

4. Type System

Not all preterms represent actor configurations. Unlike π -calculus where names denote communication channels, a name in the Actor model uniquely denotes a persistent agent. To capture this object paradigm we need to impose a certain discipline on the use of names, which we do using a type system. Well-typed preterms, called terms, will represent actor configurations.

Strictly enforcing all actor properties would make $\Lambda\pi$ too weak to express certain communication patterns. One such scenario is where, instead of assuming a new behavior immediately after receiving a message (as required by persistence property), an actor has to wait until certain synchronization conditions are met before processing the next message. For example, such a delaying mechanism is required to express polyadic communication, where an actor has to delay the assumption of a behavior and processing of other messages until all the arguments are transferred. We therefore relax the persistence requirement, and allow actors to temporarily assume a series of fresh names, one at a time, and resume the old name at a later point. Basically, the synchronization task is delegated from one new name to another until the last one releases the actor after certain synchronization conditions are met.

A typing judgment is of the form $\rho; f \vdash C$, where ρ is the set of free names in C that denote actors in C , and $f : \rho \rightarrow \rho^*$ is a function that relates actors in C to the temporary names they have assumed currently. Specifically, $f(x) = \perp$ means that x is a regular actor name and not a temporary one, $f(x) = *$ means x is the temporary name of an actor with a private name (bound by a restriction), and $f(x) = y \notin \{\perp, *\}$ means that actor y has assumed the temporary name x . The function f has the following properties: for all $x, y \in \rho$, $f(x) \neq x$, $f(x) = f(y) \notin \{\perp, *\}$ implies $x = y$, and $f^*(f(x)) = \perp$. While the first property is obvious, the second states that an actor cannot assume more than one temporary name at the same time, and the third states that temporary names are not like regular actor names in that they themselves cannot temporarily assume new names but can only delegate their capability of releasing the original actor to new names.

We define the following functions and relations that will be used in defining the type rules.

Definition 1 Let $f_1 : \rho_1 \rightarrow \rho_1^*$ and $f_2 : \rho_2 \rightarrow \rho_2^*$.

1 We define $f_1 \oplus f_2 : \rho_1 \cup \rho_2 \rightarrow (\rho_1 \cup \rho_2)^*$ as

$$(f_1 \oplus f_2)(x) = \begin{cases} f_1(x) & \text{if } x \in \rho_1, \text{ and } f_1(x) \neq \perp \text{ or } x \notin \rho_2 \\ f_2(x) & \text{otherwise} \end{cases}$$

Note that \oplus is associative.

2 If $\rho \subset \rho_1$ we define $f|\rho : \rho \rightarrow \rho^*$ as

$$(f|\rho)(x) = \begin{cases} * & \text{if } f(x) \in \rho_1 - \rho \\ f(x) & \text{otherwise} \end{cases}$$

3 We say f_1 and f_2 are compatible if $f = f_1 \oplus f_2$ has following properties: $f = f_2 \oplus f_1$, and for all $x, y \in \rho_1 \cup \rho_2$, $f(x) \neq x$, $f^*(f(x)) = \perp$, and $f(x) = f(y) \notin \{\perp, *\}$ implies $x = y$. \square

Definition 2 For a tuple \tilde{x} , we define $ch(\tilde{x}) : \{\tilde{x}\} \rightarrow \{\tilde{x}\}^*$ as $ch(\epsilon) = \{\}$, and if $len(x) = n$, $ch(\tilde{x})(x_i) = x_{i+1}$ for $1 \leq i < n$ and $ch(\tilde{x})(x_n) = \perp$. \square

Table 1. Type rules for $A\pi$.

<i>NIL</i> : $\emptyset; \{\} \vdash 0$	<i>MSG</i> : $\emptyset; \{\} \vdash \bar{x}y$
<i>ACT</i> : $\frac{\rho_i; f \vdash C}{\{x\} \cup \hat{z}; ch(x, \hat{z}) \vdash x(y).C}$	<i>if</i> $\rho - \{x\} = \hat{z}$, $y \notin \rho$, and $f = \begin{cases} ch(x, \hat{z}) & \text{if } x \in \rho \\ ch(\epsilon, \hat{z}) & \text{otherwise} \end{cases}$
<i>COND</i> : $\frac{\rho_1; f_1 \vdash C_1 \quad \rho_2; f_2 \vdash C_2}{\rho_1 \cup \rho_2; f_1 \oplus f_2 \vdash [x = y](C_1, C_2)}$	<i>if</i> f_1 and f_2 are compatible
<i>COMP</i> : $\frac{\rho_1; f_1 \vdash C_1 \quad \rho_2; f_2 \vdash C_2}{\rho_1 \cup \rho_2; f_1 \oplus f_2 \vdash C_1 C_2}$	<i>if</i> $\rho_1 \cap \rho_2 = \emptyset$
<i>RES</i> : $\frac{\rho; f \vdash C}{\rho - \{x\}; f \rho - \{x\} \vdash (\nu x)C}$	
<i>INST</i> : $\{\tilde{x}\}; ch(\tilde{x}) \vdash B(\tilde{x}; \tilde{y})$	<i>if</i> $len(\tilde{x}) = 2$ implies $x_1 \neq x_2$

The type rules are shown in Table 1. Rules *NIL* and *MSG* are obvious. In the *ACT* rule, if $\hat{z} = \{z\}$ then actor z has assumed temporary name x . The condition $y \notin \rho$ ensures that actors are not created with names received in a message. In the terminology of [14], only output capability of names can be passed in messages. The conditions $y \notin \rho$ and $\rho - \{x\} = \hat{z}$ together guarantee the freshness property by ensuring that new actors are created with fresh names. Note that it is possible for x to be a regular name, i.e. $\rho - \{x\} = \emptyset$, and disappear after receiving a message, i.e. $x \notin \rho$. We interpret this as the actor x assuming a *Sink* behavior that simply consumes all messages it receives. With this interpretation the persistence property is not violated.

The compatibility check in *COND* rule prevents errors such as two actors, each in a different branch, assuming the same temporary name, or the same actor assuming different temporary names in different branches. The *COMP* rule guarantees the uniqueness property by ensuring that the two composed configurations do not contain actors with the same name. In the *RES* rule, f is updated so that if x has assumed a temporary name y in C , then y 's role as a temporary name is remembered but x is forgotten. The *INST* rule assumes that if $len(\tilde{x}) = 2$ then $B\langle\tilde{x};\tilde{y}\rangle$ denotes an actor x_2 that has assumed temporary name x_1 .

Type checking a preterm involves checking the accompanying behavior definitions. For *INST* rule to be sound, for every definition $B \stackrel{def}{=} (\tilde{x};\tilde{y})x_1(z).C$ and substitution $\sigma = \{\tilde{u}, \tilde{v}/\tilde{x}, \tilde{y}\}$ that is one-to-one on $\{\tilde{x}\}$, the judgment $\{\tilde{u}\}; ch(\tilde{u}) \vdash (x_1(z).C)\sigma$ should be derivable. From Lemma 2, it follows that this constraint is satisfied if $\{\tilde{x}\}; ch(\tilde{x}) \vdash x_1(z).C$ is derivable. Thus, a preterm is well-typed only if for each accompanying behavior definition $B \stackrel{def}{=} (\tilde{x};\tilde{y})x_1(z).C$, the judgment $\{\tilde{x}\}; ch(\tilde{x}) \vdash x_1(z).C$ is derivable.

The following lemma states a soundness property of the type system.

Lemma 1 *If $\rho; f \vdash C$ then $\rho \subset fn(C)$, and for all $x, y \in \rho$, $f(x) \neq x$, $f^*(f(x)) = \perp$, and $f(x) = f(y) \notin \{\perp, *\}$ implies $x = y$. Further, if $\rho'; f' \vdash C$ then $\rho = \rho'$ and $f = f'$. \square*

Not all substitutions on a term C yield terms. A substitution σ may identify distinct actor names in C and therefore violate the uniqueness property. But, if σ renames different actors in C to different names then $C\sigma$ will be well typed.

Lemma 2 *If $\rho; f \vdash C$ and σ is one-to-one on ρ then $\sigma(\rho); f\sigma \vdash C\sigma$. \square*

5. Reduction Semantics

Reduction semantics of $\Lambda\pi$ is the same as that of π -calculus with mismatch. It is defined in terms of the usual structural congruence over preterms and reduction rules shown in Definition 3 and Table 2. We use \Longrightarrow to denote the reflexive transitive closure of \longrightarrow .

Definition 3 (structural congruence) *The relation \equiv is the smallest congruence relation on preterms closed under the following laws:*

- 1 If $C_1 \equiv_\alpha C_2$ then $C_1 \equiv C_2$.
- 2 The combinator, $|$, is commutative and associative with 0 as identity.
- 3 $(\nu x, y)C \equiv (\nu y, x)C$, $(\nu x)0 \equiv 0$.
- 4 If $x \notin fn(C_2)$ then $(\nu x)C_1|C_2 \equiv (\nu x)(C_1|C_2)$.
- 5 If $B \stackrel{def}{=} (\tilde{x}; \tilde{y})x_1(z).C$, $len(\tilde{u}) = len(\tilde{x})$, and $len(\tilde{v}) = len(\tilde{y})$ then $B\langle\tilde{u}; \tilde{v}\rangle \equiv (x_1(z).C)\{\tilde{u}, \tilde{v}\}/\{\tilde{x}, \tilde{y}\}$. \square

Table 2. Reduction rules for $\Lambda\pi$.

$RECV: \quad x(y).C \mid \bar{x}z \longrightarrow C\{z/y\}$
$IF: \quad [x = x](C_1, C_2) \longrightarrow C_1 \quad ELSE: \quad [x = y](C_1, C_2) \longrightarrow C_2 \quad \text{if } x \neq y$
$HIDE: \quad \frac{C \longrightarrow C'}{(\nu x)C \longrightarrow (\nu x)C'} \quad PAR: \quad \frac{C_1 \longrightarrow C'_1}{C_1 C_2 \longrightarrow C'_1 C_2}$
$REQV: \quad \frac{C'_1 \longrightarrow C'_2}{C_1 \longrightarrow C_2} \quad \text{if } \begin{array}{l} C_1 \equiv C'_1 \\ C_2 \equiv C'_2 \end{array}$

Lemma 3 and Theorem 1 state that type system respects both the structural congruence and reduction rules.

Lemma 3 *Let $C_1 \equiv C_2$. Then $fn(C_1) = fn(C_2)$, and $\rho; f \vdash C_1$ if and only if $\rho; f \vdash C_2$. \square*

Theorem 1 (subject reduction 1) *Let $\rho; f \vdash C$ and $C \longrightarrow C'$. Then $\rho'; f' \vdash C'$, for some $\rho' \subset \rho$, and $f' : \rho' \rightarrow \rho'^*$ satisfying the following conditions: $f'(x) = \perp$ if $f(x) = \perp$, $f'(x) \in \{f(x), \perp\}$ otherwise. \square*

Since well-typed terms are closed under reduction, it follows that actor properties are preserved during a computation. However, note that the

source and the target of a transition need not have the same typing judgment. This is because of two reasons. First, actors may disappear. As the reader may recall, this is interpreted as the actor assuming a sink behavior. Second, an actor with a temporary name may re-assume its original name, or decide to never assume it.

We show how the ability to temporarily assume a fresh name can be used to encode polyadic communication in $\mathsf{A}\pi$. We assume that the subject of a polyadic receive is not a temporary name. In particular, in the encoding below, x cannot be a temporary name. The idea behind translation is to let x temporarily assume a fresh name z which is used to receive all the arguments without any interference from other messages, and re-assume x after the receipt. For fresh u, z we have

$$\begin{aligned}
\llbracket \bar{x}(y_1, \dots, y_n) \rrbracket &= (\nu u)(\bar{x}u \mid S_1\langle u; y_1, \dots, y_n \rangle) \\
S_i &\stackrel{\text{def}}{=} (u; y_i, \dots, y_n)u(z).(\bar{z}y_i \mid S_{i+1}\langle u; y_{i+1}, \dots, y_n \rangle) & 1 \leq i < n \\
S_n &\stackrel{\text{def}}{=} (u; y_n)u(z).\bar{z}y_n \\
\llbracket x(y_1, \dots, y_n).C \rrbracket &= x(u).(\nu z)(\bar{u}z \mid R_1\langle z, \hat{x}; u, \tilde{a} \rangle) \\
R_i &\stackrel{\text{def}}{=} (z, \hat{x}; u, \tilde{a})z(y_i).(\bar{u}z \mid R_{i+1}\langle z, \hat{x}; u, \tilde{a} \rangle) & 1 \leq i < n \\
R_n &\stackrel{\text{def}}{=} (z, \hat{x}; u, \tilde{a})z(y_n).(\bar{u}z \mid \llbracket C \rrbracket)
\end{aligned}$$

where $\tilde{a} = fn(x(y_1, \dots, y_n).C) - \{x\}$, and $\hat{x} = \{x\}$ if for some ρ, f , we have $\rho \cup \{x\}; f \vdash \llbracket C \rrbracket$, and $\hat{x} = \emptyset$ otherwise.

The formalism thus far does not account for *fairness* in message deliveries that is required by the Actor model. We do not consider fairness, as it does not make a difference to the may testing theory we are concerned with. The reader is referred to Section 9 for further discussion about this.

6. May Testing

We now instantiate the general notion of may testing [7] on $\mathsf{A}\pi$. As in any typed calculus, testing in $\mathsf{A}\pi$ takes typing into account; an observer O can be used to test C only if $C|O$ is well typed. Since the set of valid tests varies between configurations, we parameterize the may preorder with the set of observers that is used to decide the order.

Definition 4 (may testing) *Observers are actor configurations that can emit a special message $\bar{\mu}\mu$. We let O range over the set of observers. For C, O such that $C|O$ is well-typed, we say C may O if $C|O \Longrightarrow C'|\bar{\mu}\mu$ for some C' . Let $\rho_1; f_1 \vdash C_1$ and $\rho_2; f_2 \vdash C_2$. Then for ρ such that $\rho_1, \rho_2 \subset \rho$ we say $C_1 \stackrel{\text{E}}{\sim}_{\rho} C_2$ if for every O such that $\rho'; f' \vdash O$ and*

$\rho' \cap \rho = \emptyset$, C_1 may O implies C_2 may O . We say $C_1 \simeq_\rho C_2$ if $C_1 \stackrel{\text{E}}{\sim}_\rho C_2$ and $C_2 \stackrel{\text{E}}{\sim}_\rho C_1$. Note that $\stackrel{\text{E}}{\sim}_\rho$ is reflexive and transitive, and \simeq_ρ is an equivalence relation. \square

The parameter of a preorder indicates the size of the observer set that is used to decide the order; the larger the parameter, the smaller the observer set. From this observation, it is easy to see that when $\rho_1 \subset \rho_2$, we have $C_1 \stackrel{\text{E}}{\sim}_{\rho_1} C_2$ implies $C_1 \stackrel{\text{E}}{\sim}_{\rho_2} C_2$, but not the converse. To see why the converse doesn't hold, we have $0 \simeq_{\{x\}} \bar{x}x$, but only $0 \stackrel{\text{E}}{\sim}_\emptyset \bar{x}x$ and $\bar{x}x \not\stackrel{\text{E}}{\sim}_\emptyset 0$. Similarly, $\bar{x}x \simeq_{\{x,y\}} \bar{y}y$, but $\bar{x}x \not\stackrel{\text{E}}{\sim}_\emptyset \bar{y}y$ and $\bar{y}y \not\stackrel{\text{E}}{\sim}_\emptyset \bar{x}x$. However, the converse holds if $\text{fn}(C_1) \cup \text{fn}(C_2) \subset \rho_1$.

Theorem 2 *Let $\rho_1 \subset \rho_2$. Then $C_1 \stackrel{\text{E}}{\sim}_{\rho_1} C_2$ implies $C_1 \stackrel{\text{E}}{\sim}_{\rho_2} C_2$. Further, if $\text{fn}(C_1) \cup \text{fn}(C_2) \subset \rho_1$ then $C_1 \stackrel{\text{E}}{\sim}_{\rho_2} C_2$ implies $C_1 \stackrel{\text{E}}{\sim}_{\rho_1} C_2$. \square*

7. Labeled Transition System

We give an alternate characterization of may testing which does not involve quantification over observing contexts. The characterization is trace-based, i.e. it is in terms of sequences of observable actions, namely the message exchanges, that a configuration may perform while interacting with its environment.

The set of possible message exchanges at any time is determined by the current ownership of names, i.e. which names denote actors in the configuration and which those in the environment. The configuration can input only messages targeted to one of its actors that is not hidden from the environment (a receptionist), and can emit only messages targeted to an actor in the environment (an external actor). We call this the *encapsulation* property. Note that the information about ownership of names is in general not contained in the syntax of a configuration as internal actors may disappear (assume sink behavior) and external names may be forgotten as the configuration evolves. We therefore define the notion of a configuration interface that records the history of ownership of names, and use it to define a labeled transition system that characterizes observable actions.

Definition 5 (interfaces) *An interface is a pair of sets of names written as $[\rho, \chi]$, where $\rho \cap \chi = \emptyset$. We let I range over interfaces. We define an ordering on interfaces as $[\rho_1, \chi_1] \leq [\rho_2, \chi_2]$ if $\rho_1 \subset \rho_2$ and $\chi_1 \subset \chi_2 \cup \rho_2$. \square*

Lemma 4 *The relation \leq on interfaces is a partial order. \square*

We associate a configuration with interface $[\rho, \chi]$ to mean that names in ρ denote receptionists of the configuration and those in χ denote its external actors. Thus, the configuration can input only messages with target in ρ and emit messages with target in χ . Note that since the computational history of a configuration is not contained in its syntax, a configuration can have several possible interfaces. The idea behind partial order on interfaces is that if $I_1 \leq I_2$ and I_1 is a possible interface of a configuration then so is I_2 .

Definition 6 *Let $\rho; f \vdash C$, and $\chi = fn(C) - \rho$. Then we say $[\rho', \chi']$ is a possible interface of C and write $C : [\rho', \chi']$ if $[\rho, \chi] \leq [\rho', \chi']$. We call $[\rho, \chi]$ the minimal interface of C .*

Remark: *Note that as a direct consequence of Lemma 3, if $C_1 \equiv C_2$ then $C_1 : [\rho, \chi]$ if and only if $C_2 : [\rho, \chi]$. \square*

We define labeled transitions over configurations with interfaces, which are written as $\langle\langle C \rangle\rangle_\chi^\rho$. We say $\langle\langle C \rangle\rangle_\chi^\rho$ is well-formed if and only if $C : [\rho, \chi]$. The transition rules are given in Table 3. Transition labels can be of five forms: τ (a silent action), $\bar{x}y$ (free output of a message with target x and content y), $\bar{x}(y)$ (bound output), xy (free input of a message) and $x(y)$ (bound input). We denote the set of all visible actions (non- τ) actions by \mathcal{L} , and let α range over \mathcal{L} . The functions $fn(\cdot)$, $bn(\cdot)$ and $n(\cdot)$ are defined on \mathcal{L} the usual way. To have a convenient uniform notation for free and bound actions we use the following convention: $(\emptyset)\bar{x}y = \bar{x}y$, $(\{y\})\bar{x}y = \bar{x}(y)$, and similarly for input actions. We define a complementation function on \mathcal{L} as $(\hat{y})xy = (\hat{y})\bar{x}y$, $(\hat{y})\bar{x}y = (\hat{y})xy$.

Table 3. Labelled transition system for $A\pi$

<i>IN:</i>	$\langle\langle C \rangle\rangle_\chi^\rho \xrightarrow{(\hat{y})xy} \langle\langle C \mid \bar{x}y \rangle\rangle_{(\chi \cup \{y\})-\rho}^\rho$	if $\hat{y} \cap (\rho \cup \chi) = \emptyset$, $x \in \rho$
<i>OUT:</i>	$\langle\langle (\nu \hat{y})(C \mid \bar{x}y) \rangle\rangle_\chi^\rho \xrightarrow{(\hat{y})\bar{x}y} \langle\langle C \rangle\rangle_\chi^{\rho \cup \hat{y}}$	if $\hat{y} \cap (\rho \cup \chi) = \emptyset$, $x \in \chi$
<i>TAU:</i>	$\frac{C \longrightarrow C'}{\langle\langle C \rangle\rangle_\chi^\rho \xrightarrow{\tau} \langle\langle C' \rangle\rangle_\chi^\rho}$	<i>LEQV:</i> $\frac{\langle\langle C'_1 \rangle\rangle_\chi^\rho \xrightarrow{\alpha} \langle\langle C'_2 \rangle\rangle_\chi^\rho}{\langle\langle C_1 \rangle\rangle_\chi^\rho \xrightarrow{\alpha} \langle\langle C_2 \rangle\rangle_\chi^\rho}$ if $C_1 \equiv C'_1$ $C_2 \equiv C'_2$

The labeled transition system is essentially a simple extension of the reduction system to include observable actions. The *IN* and *OUT* rules together capture the encapsulation property we described earlier. The *IN* rule states that a configuration can receive only a message targeted to one of its receptionists. The message is asynchronous and is added to

the pool of messages in the configuration. The external actor set of the interface may expand as the received message may contain new external actor names. The *OUT* rule states that only messages targeted to an external actor can leave the configuration. The receptionist set may expand because names of hidden actors can be exported in the message. Note that fresh names are chosen for these new receptionists so that uniqueness property is preserved.

Lemma 5 states that the transition system is consistent with the reduction system. Theorem 3 states the soundness of the transition system and also characterizes the evolution of configuration interfaces.

Lemma 5 *If $C : [\rho, \chi]$ then $C \longrightarrow C'$ if and only if $\langle\langle C \rangle\rangle_{\chi}^{\rho} \xrightarrow{\tau} \langle\langle C' \rangle\rangle_{\chi}^{\rho}$.*
□

Theorem 3 (subject reduction 2) *If $C_1 : [\rho_1, \chi_1]$ and $\langle\langle C_1 \rangle\rangle_{\chi_1}^{\rho_1} \xrightarrow{\alpha} \langle\langle C_2 \rangle\rangle_{\chi_2}^{\rho_2}$ then $C_2 : [\rho_2, \chi_2]$, $\rho_1 \subset \rho_2$ and $\chi_1 \subset \chi_2$.*

Remark: *Note that $[\rho_1, \chi_1] \leq [\rho_2, \chi_2]$.* □

We let s, r, t range over \mathcal{L}^* . For $s = \alpha_1 \dots \alpha_i \dots \alpha_n$, we define $len(s) = n$, and $s(i) = \alpha_i$, for $1 \leq i \leq len(s)$. The functions $fn(\cdot)$, $bn(\cdot)$ and $n(\cdot)$ are defined on \mathcal{L}^* the obvious way. The complementation function on \mathcal{L} is extended to \mathcal{L}^* the obvious way. We use \Longrightarrow to denote the reflexive transitive closure of $\xrightarrow{\tau}$, and $\xRightarrow{\alpha}$ to denote $\Longrightarrow \xrightarrow{\alpha} \Longrightarrow$. Note that \Longrightarrow is overloaded to denote both sequences of reductions and $\xrightarrow{\tau}$ transitions, but its context of use will always clarify which one is being used. For $s = l.s'$ we use $\langle\langle C_1 \rangle\rangle_{\chi_1}^{\rho_1} \xrightarrow{s} \langle\langle C_2 \rangle\rangle_{\chi_2}^{\rho_2}$ to denote $\langle\langle C_1 \rangle\rangle_{\chi_1}^{\rho_1} \xrightarrow{l} \xrightarrow{s'} \langle\langle C_2 \rangle\rangle_{\chi_2}^{\rho_2}$, and similarly $\langle\langle C_1 \rangle\rangle_{\chi_1}^{\rho_1} \xRightarrow{s} \langle\langle C_2 \rangle\rangle_{\chi_2}^{\rho_2}$ to denote $\langle\langle C_1 \rangle\rangle_{\chi_1}^{\rho_1} \xRightarrow{l} \xRightarrow{s'} \langle\langle C_2 \rangle\rangle_{\chi_2}^{\rho_2}$. We write $\langle\langle C \rangle\rangle_{\chi}^{\rho} \xRightarrow{s}$ if $\langle\langle C \rangle\rangle_{\chi}^{\rho} \xRightarrow{s} \langle\langle C' \rangle\rangle_{\chi'}^{\rho'}$ for some C', ρ', χ' , and similarly for $\langle\langle C \rangle\rangle_{\chi}^{\rho} \xrightarrow{s}$ and $\langle\langle C \rangle\rangle_{\chi}^{\rho} \xrightarrow{\tau}$.

The sequences of observable actions, called *interaction paths*, that a configuration C with interface $[\rho, \chi]$ may perform are precisely $s \in \mathcal{L}^*$ such that $\langle\langle C \rangle\rangle_{\chi}^{\rho} \xRightarrow{s}$. The following lemma, which is true in $A\pi$, relates a computation involving two composed configurations to the interaction paths that each exhibits during the computation.

Lemma 6 (zip-unzip) *Let $C_1 : [\rho_1, \chi_1], C_2 : [\rho_2, \chi_2]$, and $\rho_1 \cap \rho_2 = \emptyset$. Then $C_1|C_2 \Longrightarrow C$ if and only if for some s , $\langle\langle C_1 \rangle\rangle_{\chi_1}^{\rho_1} \xRightarrow{s} \langle\langle C'_1 \rangle\rangle_{\chi'_1}^{\rho'_1}$, $\langle\langle C_2 \rangle\rangle_{\chi_2}^{\rho_2} \xRightarrow{\bar{s}} \langle\langle C'_2 \rangle\rangle_{\chi'_2}^{\rho'_2}$ and $C \equiv (\nu \bar{z})(C'_1|C'_2)$, where $\{\bar{z}\} = bn(s)$.* □

8. Alternate Characterization of May Testing

We present an alternate characterization of may testing in $A\pi$ that is based on interaction paths. We follow the same approach used for

asynchronous π -calculus by Boreale [11]. However, differences between the two calculi, such as in name matching capabilities and notions of observability, lead to changes in the characterization and require different proofs to establish it. To demonstrate these differences, we follow Boreale’s proof layout and highlight the differences as they arise.

Definition 7 and Lemma 7 demonstrate the relation between interaction paths exhibited by a configuration and the evolution of its interface.

Definition 7 *We define the following functions on interaction paths*

$$\begin{aligned} rcp([\rho, \chi], \epsilon) &= \rho & rcp([\rho, \chi], s.(\hat{y})\bar{x}y) &= \hat{y} \cup rcp([\rho, \chi], s) \\ rcp([\rho, \chi], s.(\hat{y})xy) &= rcp([\rho, \chi], s) \\ ext([\rho, \chi], \epsilon) &= \chi & ext([\rho, \chi], s.(\hat{y})\bar{x}y) &= ext([\rho, \chi], s) \\ ext([\rho, \chi], s.(\hat{y})xy) &= (\{y\} \cup ext([\rho, \chi], s)) - rcp([\rho, \chi], s) \quad \square \end{aligned}$$

Lemma 7 *If $\langle\langle C \rangle\rangle_{\chi}^{\rho} \xrightarrow{s} \langle\langle C' \rangle\rangle_{\chi'}^{\rho'}$ then*

- 1 $\rho' = rcp([\rho, \chi], s)$ and $\chi' = ext([\rho, \chi], s)$,
- 2 $s = s_1.(\hat{y})xy.s_2$ implies $x \in rcp([\rho, \chi], s_1)$, and $\hat{y} \cap (rcp([\rho, \chi], s_1) \cup ext([\rho, \chi], s_1)) = \emptyset$.
- 3 $s = s_1.(\hat{y})\bar{x}y.s_2$ implies $x \in ext([\rho, \chi], s_1)$, and $y \in rcp([\rho, \chi], s_1) \cup ext([\rho, \chi], s_1)$ if and only if $\hat{y} = \emptyset$.
- 4 $rcp([\rho, \chi], s) \cup ext([\rho, \chi], s) = n(s) \cup \rho \cup \chi$, and
- 5 $s = s_1.\alpha.s_2$ implies $bn(\alpha) \cap (n(s_1) \cup \rho \cup \chi) = \emptyset$. \square

For $\rho \cap \chi = \emptyset$, we define $\mathcal{L}^*[\rho, \chi]$ as the set of all $s \in \mathcal{L}^*$ that satisfy conditions 2 and 3 of Lemma 7. We define alpha equivalence on paths the obvious way, and work modulo alpha equivalence. Note that $\mathcal{L}^*[\rho, \chi]$ is not closed under alpha renaming, that is for $s \in \mathcal{L}^*[\rho, \chi]$ there may be $r \equiv_{\alpha} s$ but $r \notin \mathcal{L}^*[\rho, \chi]$. Therefore, we will only consider alpha renaming that does not result in such ill-formed paths

Definition 8 (path transformation) *We define a relation \preceq on \mathcal{L}^* as the reflexive transitive closure of the relation defined in Table 4. \square*

The intuition behind laws in Table 4, which is stated formally in Lemma 8, is that, for $r, s \in \mathcal{L}^*[\rho, \chi]$ and $r \prec s$, if a configuration exhibits \bar{s} then it can also exhibit \bar{r} . *L3* states that two consecutive outputs can be commuted, while *L4* states that two consecutive inputs can be commuted. *L5* states that an output can be postponed to after an input, provided the input doesn’t use the bound name exported by the output. *L3* and *L5* can be used to postpone outputs, and *L4* and *L5* to prepone

Table 4. A relation on interaction paths. In **L3** and **L4**, $M = \hat{v}$, $N = \hat{y}$ if $v \notin \hat{y}$, and $M = \hat{y}$, $N = \emptyset$ otherwise.

(L1)	$s.(\hat{y})\bar{x}y$	\prec	s
(L2)	s	\prec	$s.(\hat{y})xy$
(L3)	$s_1.(M)uv.(N)xy.s_2$	\prec	$s_1.(\hat{y})xy.(\hat{v})uv.s_2$
(L4)	$s_1.(M)\bar{u}v.(N)\bar{x}y.s_2$	\prec	$s_1.(\hat{y})\bar{x}y.(\hat{v})\bar{u}v.s_2$
(L5)	$s_1.(\hat{v})\bar{u}v.(\hat{y})xy.s_2$	\prec	$s_1.(\hat{y})xy.(\hat{v})\bar{u}v.s_2$ if $u, v \notin \hat{y}$

inputs. These two rules capture the essence of asynchrony. *L1* states that additional inputs may be appended, and *L2* states that a tailing output can be removed as there is no interaction after it that depends on it.

Lemma 8 *If $\langle\langle C \rangle\rangle_\chi^\rho \xrightarrow{\bar{s}}$, $r \preceq s$ and $\bar{r} \in \mathcal{L}^*[\rho, \chi]$, then $\langle\langle C \rangle\rangle_\chi^\rho \xrightarrow{\bar{r}}$. \square*

We now compare our laws with those of Boreale. The mismatch capability in $\mathbf{A}\pi$ enables distinguishing bound names from free names. Thus, Boreale's law that allows replacing bound names in an input action with free names is not applicable in $\mathbf{A}\pi$. Furthermore, Boreale's *annihilation* law, which states that a configuration can consume a pair of complementary interactions, is not needed in $\mathbf{A}\pi$ for two reasons. First, due to the encapsulation property a configuration can never exhibit complementary actions. Second, because we do not have a law that substitutes bound names with fresh names, no path with complementary actions is related to a path in $\mathcal{L}^*[\rho, \chi]$. Finally, as opposed to asynchronous inputs allowed by the *IN* rule, Boreale's LTS uses synchronous inputs. As a consequence, *L4* is not applicable there. These differences lead to a stronger characterization of may preorder for $\mathbf{A}\pi$ (see below).

Using \preceq we define the following preorder on configurations, which we will prove to be an alternate characterization of may preorder.

Definition 9 *Let $[\rho_1, \chi_1]$ be the minimal interface of C_1 and $[\rho_2, \chi_2]$ that of C_2 . For ρ such that $\rho_1, \rho_2 \subset \rho$, we say $C_1 \ll_\rho C_2$ if for $\chi = (\chi_1 \cup \chi_2) - \rho$, $\langle\langle C_1 \rangle\rangle_\chi^\rho \xrightarrow{s}$ implies $\langle\langle C_2 \rangle\rangle_\chi^\rho \xrightarrow{r}$ for some $r \preceq s$. \square*

Although it is easy to see that $C_1 \ll_\rho C_2$ implies $C_1 \stackrel{\Xi}{\sim}_\rho C_2$, the reverse direction is more involved. To prove the reverse direction, we construct for a given $s \in \mathcal{L}^*[\rho, \chi]$, an observer O such that for $C : [\rho, \chi]$, if $C \underline{\text{may}} O$ then $\langle\langle C \rangle\rangle_\chi^\rho \xrightarrow{r}$ for some $r \preceq s$.

Definition 10 (canonical observer) For $s \in \mathcal{L}^*[\rho, \chi]$, we define an observer

$$O([\rho, \chi], s) = (\nu \tilde{x}, z) (|_{x_i \in \text{ext}([\rho, \chi], s)} \text{Proxy}(s, x_i, z) \mid O'([\rho, \chi], s, z)),$$

where for u, v fresh

$$\{\tilde{x}\} = \text{bn}(s) - \text{rcp}([\rho, \chi], s)$$

$$O'([\rho, \chi], \epsilon, z) \triangleq \bar{\mu}$$

$$O'([\rho, \chi], (\hat{y})xy.s, z) \triangleq \bar{x}y | O'([\rho, \chi \cup \{y\} - \rho], s, z)$$

$$O'([\rho, \chi], \bar{x}y.s, z) \triangleq z(u, v).[u = x \wedge v = y](O'([\rho, \chi], s, z), 0)$$

$$O'([\rho, \chi], \bar{x}(y).s, z) \triangleq z(u, y).[u = x \wedge y \notin (\rho \cup \chi)](O'([\rho \cup \{y\}, \chi], s, z), 0)$$

$$\text{Proxy}(\epsilon, x, z) = 0$$

$$\text{Proxy}((\hat{y})xy.s, x, z) \triangleq \text{Proxy}(s, x, z)$$

$$\text{Proxy}((\hat{y})\bar{x}y.s, x, z) \triangleq x(v).(\bar{z}\langle x, v \rangle \mid \text{Proxy}(s, x, z))$$

In the above, \triangleq is used for macro definitions. \square

The observer $O([\rho, \chi], s)$ consists of a collection of proxies and a central matcher. There is one forwarding proxy for each external name a configuration C knows while doing s . This forwarding mechanism, which is absent in Boreale's construction, is essential in our case because of uniqueness of actor names. The matcher which analyzes the forwarded messages, keeps track of names in the "current" interface of C and uses them to distinguish bound names from free names in outputs. This technique works because if $(\hat{y})\bar{x}y.s \in \mathcal{L}^*[\rho, \chi]$ and $y \notin \rho \cup \chi$ then $\hat{y} = \{y\}$. The abbreviations \notin and \wedge used in the definition can be encoded using the conditional construct. The encoding of \notin requires the ability to mismatch names. Note that the definition also uses polyadic communication between proxies and matcher, whose encoding was shown in Section 5.

We not only require a different construction for the canonical observer than Boreale's, but also an essentially different argument for establishing Lemma 9 (see Appendix for proof). For $s \in \mathcal{L}^*[\rho, \chi]$, let $\text{ffi}([\rho, \chi], s)$ be the set of all names y such that s can be written as $s_1.xy.s_2$ and $y \notin \text{rcp}([\rho, \chi], s_1) \cup \text{ext}([\rho, \chi], s_1)$. It is easy to show that if $s \in \mathcal{L}^*[\rho, \chi]$ and $\chi' = \chi \cup \text{ffi}([\rho, \chi], s)$, then $O([\rho, \chi], s) : [\chi', \rho]$.

Lemma 9 Let $r, s \in \mathcal{L}^*[\rho, \chi]$ and $\chi' = \chi \cup \text{ffi}([\rho, \chi], s)$. Then

$$\langle\langle O([\rho, \chi], s) \rangle\rangle_{\rho}^{\chi'} \xrightarrow{\bar{r}. \bar{\mu}} \text{implies } r \preceq s. \quad \square$$

Following is the alternate characterization of may preorder.

Theorem 4 $C_1 \stackrel{\Xi}{\sim}_{\rho} C_2$ if and only if $C_1 \ll_{\rho} C_2$. \square

This alternate characterization can be further strengthened to set inclusion in the case of $A\pi$. This is a consequence of Lemma 10 which is not true in Boreale's setting. Note that Theorem 5 renders the interaction-path preorder to a proof tool rather than a part of the characterization.

Lemma 10 *Let $r, s \in \mathcal{L}^*[\rho, \chi]$. Then $r \prec s$ implies $\bar{s} \prec \bar{r}$. \square*

Theorem 5 *Let $[\rho_1, \chi_1]$ be the minimal interface of C_1 and $[\rho_2, \chi_2]$ that of C_2 . For ρ such that $\rho_1, \rho_2 \subset \rho$, if $C_1 \ll_{\rho} C_2$ and $\chi = (\chi_1 \cup \chi_2) - \rho$, then $\langle\langle C_1 \rangle\rangle_{\chi}^{\rho} \xrightarrow{s} \implies \langle\langle C_2 \rangle\rangle_{\chi}^{\rho} \xrightarrow{s}$. \square*

9. Discussion and Related Work

A possible direction of future work is to give a complete axiomatization for finite configurations, i.e. configurations that do not use recursive behavior definitions. Preliminary results for characterization of may testing for variants of $A\pi$ with different name matching capabilities have been given in [18]. These results have also been extended in [17] to get a characterization for $L\pi$ [12].

We have not considered fairness property of the Actor model in this paper as it does not affect the notion of may testing. May testing is concerned only with the occurrence of an event after a finite computation, while fairness requires eventual delivery of messages, thereby affecting only potentially infinite computations. An interesting consequence of fairness is that must equivalence implies may equivalence, which was shown for a specific Actor based language in [2]. It can be shown by a similar argument that this result holds in $A\pi$ also.

Several calculi [5, 6, 10, 15] and programming languages [2, 5] have been proposed for actors. Since these works were motivated by different reasons, such as design of high-level languages or type systems for certain generic problems in object-oriented languages, their systems are not faithful to the pure Actor model [1]. For instance, they either are equipped with high level programming constructs that are not intrinsic to actors, or ignore actor properties such as uniqueness and persistence. Furthermore, these systems are not directly comparable to π -calculus. In contrast, our aim was to investigate a theory for the pure Actor model and compare it with that of Asynchronous π -calculus.

Notions of equivalence and semantic models have been studied for actors, such as asynchronous bisimulation [6], testing equivalences [2], event diagrams [4], and interaction paths [16]. We have not only related may testing [2] to the interaction paths model [16], but also related our characterizations to that of asynchronous π -calculus given in [11].

References

- [1] G. Agha. *Actors: A Model of Concurrent Computation in Distributed Systems*. MIT Press, 1986.
- [2] G. Agha, I. A. Mason, S. F. Smith, and C. L. Talcott. A foundation for actor computation. *Journal of Functional Programming*, 7:1–72, 1997.
- [3] G. Boudol. Asynchrony and the π -Calculus. Technical Report 1702, Department of Computer Science, Inria Univeristy, May 1992.
- [4] W.D. Clinger. *Foundations of Actor Semantics*. PhD thesis, Massachusetts Institute of Technology, AI Laboratory, 1981.
- [5] F.Dagnat, M.Pantel, M.Colin, and P.Sallé. Typing concurrent objects and actors. In *L’Objet – Methodes formelles pour les objets (L’OBJET)*, volume 6, pages 83–106, 2000.
- [6] M. Gaspari and G. Zavattaro. An Algebra of Actors. Technical Report UBLC97-4, Department of Computer Science, Univeristy of Bologna (Italy), May 1997.
- [7] M. Hennessy. *Algebraic Theory of Processes*. MIT Press, 1988.
- [8] K. Honda and M. Tokoro. An Object Calculus for Asynchronous Communication. In *Fifth European Conference on Object-Oriented Programming*, July 1991. LNCS 512, 1991.
- [9] J-L.Colaço, M.Pantel, F.Dagnat, and P.Sallé. Safety analysis for non-uniform service availability in actors. In *Formal Methods for Open Object-based Distributed Systems*, February 1999.
- [10] J-L.Colaço, M.Pantel, and P.Sallé. Analyse de linéarité par typage dans un calcul d’acteurs primitifs. In *Actes des Journées Francophones des Langages Applicatifs (JFLA)*, 1997.
- [11] R. Pugliese M. Boreale, R. De Nicola. A theory of may testing for asynchronous languages. In *Foundations of Software Science and Computation Structures*, pages 165–179, 1999. LNCS 1578.
- [12] M. Merro and D. Sangiorgi. On Asynchrony in Name-Passing Calculi. In *Proceeding of ICALP ’98*. Springer-Verlag, 1998. LNCS 1443.
- [13] R. Milner, J. Parrow, and D. Walker. A calculus of Mobile Processes, Part I. Technical Report ECS-LFCS-89-85, Department of Computer Science, Edinburgh University, June 1989.
- [14] B. C. Pierce and D. Sangiorgi. Typing and Subtyping for Mobile Processes. *Journal of Mathematical Structures in Computer Science*, 6(5):409–454, 1996.
- [15] A. Ravara and V. Vasconcelos. Typing non-uniform concurrent objects. In *CONCUR*, pages 474–488, 2000. LNCS 1877.
- [16] C. Talcott. Composable semantic models for actor theories. *Higher-Order and Symbolic Computation*, 11(3), 1998.
- [17] Prasanna Thati, Reza Ziaei, and Gul Agha. An alternate characterization of may testing for $L\pi$. Technical Report UIUCDCS-R-2001-2256, Department of Computer Science, University of Illinois at Urbana Champaign, 2001.
- [18] Prasanna Thati, Reza Ziaei, and Gul Agha. A theory of may testing for $A\pi$. Technical Report UIUCDCS-R-2001-2207, Department of Computer Science, University of Illinois at Urbana Champaign, 2001.