A Process-Algebraic Language for Probabilistic I/O Automata[⋆]

Eugene W. Stark, Rance Cleaveland, Scott A. Smolka[⋆⋆]

Department of Computer Science, State University of New York at Stony Brook,
Stony Brook, NY 11794 USA

Abstract. We present a process-algebraic language for Probabilistic I/O Automata (PIOA). To ensure that PIOA specifications given in our language satisfy the "input-enabled" property, which requires that all input actions be enabled in every state of a PIOA, we augment the language with a set of *type inference rules*. We also equip our language with a formal operational semantics defined by a set of *transition rules*. We present a number of results whose thrust is to establish that the typing and transition rules are sensible and interact properly. The central connection between types and transition systems is that if a term is well-typed, then in fact the associated transition system is input-enabled. We also consider two notions of equivalence for our language, *weighted bisimulation equivalence* and *PIOA behavioral equivalence*. We show that both equivalences are substitutive with respect to the operators of the language, and note that weighted bisimulation equivalence is a strict refinement of behavioral equivalence.

Keywords: stochastic process algebras; typing systems and algorithms; process equivalences; continuous-time Markov chains

1 Introduction

In previous work [WSS94,WSS97] we introduced *probabilistic I/O automata* (PIOA) as a formal model for systems that exhibit concurrent and probabilistic behavior. PIOA extend the well-known I/O automaton model for nondeterministic computation [LT87] with two kinds of performance information: probability distributions representing the relative likelihood with which transitions from a given state labeled by the same input are performed; and rate information describing how long, on average, the automaton will remain in a state before taking a particular output or internal transition.

[⋆] This research was supported in part by the National Science Foundation under Grant CCR-9988155 and the Army Research Office under Grants DAAD190110003 and DAAD190110019. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation, the Army Research Office, or other sponsors.

[⋆⋆] Authors' E-mail addresses: {stark,rance,sas}@cs.sunysb.edu

PIOA are similar in many respects to *stochastic automata* [Buc99,PA91], and like stochastic automata, PIOA are associated with *continuous-time Markov chains (CTMCs).* PIOA are also equipped with a *composition operation* by which a complex automaton can be constructed from simpler components. Both PIOA and stochastic automata can thus be seen as a formalism for describing large CTMC system models from simpler components. The composition operation for PIOA is defined in essentially the same way as for stochastic automata, however, the PIOA model draws a distinction between *input* (passive) and *output* (active) actions, and in forming the composition of automata only input/input or input/output synchronization is permitted — the output/output case is prohibited.

In [SS98] we presented algorithms for calculating certain kinds of performance parameters for systems modeled in terms of PIOA. These algorithms work in a *compositional* fashion; that is, by treating the components of a composite system in succession rather than all at once. Our implementation of these algorithms, called "PIOATool," has been integrated into the Concurrency Workbench [CPS93] (CWB), as described in [ZCS03]. The CWB provides several analysis capabilities for specifications expressed in process-algebraic language, including equivalence, preorder, and model checking. It has a retargetable front end that allows it to be applied to virtually any process-algebraic language having a formal semantics defined in the "structural operational semantics" (SOS) style. To achieve the PIOATool/CWB integration, it was necessary for us to design such a process-algebraic language for PIOA-based specifications, together with an SOS semantics for the language. This language, and associated theorems about its semantics, form the subject of the present paper.

The PIOA model exhibits certain features that differentiate it from other languages previously supported by the CWB. One such feature is the fact that each transition of a PIOA, besides being labeled by an action, is also labeled by a numeric *weight*, which can be either a probability (in the case of an input transition) or a rate (in the case of an output or internal transition). Another such feature is the so-called "input-enabled" property, which requires that all input actions be enabled in every state of a PIOA. It is the second of these features that has the most impact on the design of a process-algebraic language for PIOA, since it is necessary to ensure that the input-enabled property holds for every well-formed specification in the language. The problem is that process-algebraic specifications of desired input-enabled transition systems usually have to be built up from component specifications that are not input-enabled.

To solve the problem of guaranteeing that PIOA specifications given in our language satisfy the input-enabled property, we augment the language with a set of *type inference rules*. These rules define a set of inferable *typing judgements* of the form $t : I/J \Rightarrow O$. Such a judgement asserts that for term t, I is a set of actions for which input transitions are guaranteed to be enabled at the first step of t, J is a set of actions for which input transitions are guaranteed to be enabled at all steps of t after the first, and O is a set of actions that includes at least all the outputs that may be produced by t (but which may be larger). A closed term t is called *well-typed* if there is some typing judgement that is inferable for it. Besides enforcing input-enabledness, types are used to enforce compatibility conditions for parallel composition and they also appear as hypotheses in some of the transition rules of the language's operational semantics.

We present a number of results whose thrust is to establish that the typing and transition rules are sensible and interact properly, including a principal type theorem, a connection between the types that can be inferred for a term and the transitions that can be inferred for it, and a subject reduction theorem which establishes that well-typedness is preserved across transitions. The central connection between types and transition systems is that if a term is well-typed, then in fact the associated transition system is input-enabled.

We also define two notions of equivalence for our language, *weighted bisimulation equivalence* and *PIOA behavioral equivalence*, and investigate their properties. In particular, we observe that weighted bisimulation equivalence strictly refines behavioral equivalence (a detailed proof can be found in [Sta03]) and that both equivalences are substitutive with respect to the operators of the PIOA language.

The rest of the paper develops along the following lines. Section 2 surveys some related work by other researchers. Section 3 defines the syntax of our PIOA language. Section 4 presents the language's type-inference rules and transition rules. Section 5 gives metatheoretic results that connect the typing and transition rules. Section 6 defines the two notions of equivalence and establishes that they are substitutive. Section 7 contains our concluding remarks. Due to space limitations, all proofs are omitted.

2 Related Work

Formal languages for specifying (non-probabilistic) I/O automata have previously been proposed by several researchers. The process-algebraic languages presented in [Vaa91,DNS95] ensure input-enabledness by filling in "default transitions" for missing input transitions. In the case of [Vaa91], the default transitions are "self-loop" input transitions taken from a term to itself. In [DNS95], the default transitions lead to the "unspecified I/O automaton" Ω_S. In contrast, we have found in writing actual specifications that sometimes one wants default transitions that are self-loops and sometimes one wants default transitions that go to an error state. An automatic mechanism for filling in defaults is likely to get it wrong a significant fraction of the time, resulting in a specification language that is less transparent to the user. Thus, our language does not make any attempt to fill in default transitions, but rather it employs a notion of well-typedness of terms which guarantees that all well-typed terms are input-enabled.

Another language for describing I/O automata is the IOA language of [GL00]. IOA uses guarded-command-like "transition definitions" consisting of preconditions and effects to encode I/O automata. It also provides constructs for nondeterministic choice, composition, and action hiding. Automatic code generation from IOA specifications is also supported.

A number of process algebras capturing performance-related aspects of system behavior have been proposed in the literature; see [HH02] for a comprehensive survey. Among these, EMPA [BDG98] is perhaps most closely related to our PIOA process algebra as it makes an I/O-like master-slave distinction between "active" and "passive" actions. Active and passive actions can synchronize, with the rate of the synchronization determined by the rate of the passive action, while synchronization between active actions is disallowed. Hillston [Hil94] gives a thoughtful discussion of the issues surrounding synchronization in stochastic processes, including the role of passive and active actions. The issue has also been treated more recently by Brinksma and Hermanns [BH01].

3 Syntax

Let *Act* be a set of *actions*, and let *Var* be a set of *variables*. We use $a, b, c \ldots$ to range over *Act* and we use $X, Y, Z \ldots$ to range over *Var*. Our language has the following syntax:

$$\begin{aligned} P ::= X \;\mid\; \mathrm{nil} \;\mid\; a_{(w)}\,?\,t \;\mid\; b_{(r)}\,!\,t \;\mid\; \tau_{(r)}\cdot t \;\mid \\ t_1 + t_2 \;\mid\; t_1\,{}_{O_1}\|_{O_2}\,t_2 \;\mid\; t\,[O] \;\mid\; t\{a \leftarrow a'\} \;\mid\; \mu X.t \end{aligned}$$

The informal meaning of the various constructs is as follows:

- X is a process variable, used in forming recursive processes.
- nil denotes a process with no actions and no transitions.
- $a_{(w)}\,?\,t$ denotes an *input-prefixed* process that can perform input action $a \in$ *Act* with *weight* w and then become the process t. The weight w must be a positive real number, and it is typically a probability.
- $b_{(r)}\,!\,t$ denotes an *output-prefixed* process that can perform output action $b \in$ *Act* with *rate* r and then become process denoted by t. The rate r must be a positive real number, which (as usual for CTMC-based models) we regard as the parameter of an exponential probability distribution that describes the length of time before term $b_{(r)}\,!\,t$ will perform a transition.
- $\tau_{(r)}\cdot t$ denotes an *internal-prefixed* process that can perform an internal transition with rate r and then become the process denoted by t. Here τ is a special symbol, not in *Act*, used to indicate an internal transition. The rate r must be a positive real number, as for output prefixing.
- $t_1 + t_2$ denotes a *choice* between alternatives offered by t_1 and t_2. Choices between summands prefixed by distinct input actions are determined by the environment, and amount to a form of external nondeterminism. Choices between summands prefixed by the same input action are probabilistic choices governed by the relative weights appearing in the prefixes. Choices between summands prefixed by output or internal actions are probabilistic choices governed by the usual *race condition* involving the rates appearing in the prefixes. Choices between input-prefixed summands and summands prefixed by output or internal actions are ultimately resolved by a race between the process and its environment.

- $t_1 \,{}_{O_1}\|_{O_2}\, t_2$ denotes a process that is the parallel composition of the processes denoted by t_1 and t_2. The sets O_1 and O_2 are the sets of output actions controlled by t_1 and t_2, respectively. These sets are required to be disjoint.
- $t\,[O]$ denotes a term t in which all output transitions labeled by actions not in the set O have been *hidden* by transforming them into internal transitions.
- $t\{e \leftarrow e'\}$ denotes the *renaming* of action e to e' in t. The typing rules presented below will ensure that action e' is a fresh action that is not already an input or output action for t.
- $\mu X.t$ denotes a recursively defined process in the usual way. The recursion variable X is required to be be *guarded* by input, output, or internal prefixing in the expression t.

4 Semantics

4.1 Types

Our language is equipped with a set of inference rules for inferring *typing judgements*, which take the form $t : I/J \Rightarrow O$ where I, J, and O are sets of actions. The intuitive meaning of such judgements was described in Section 1. We use the abbreviation $I \Rightarrow O$ for the special case $I/I \Rightarrow O$ in which the sets I and J are equal. A closed term t is *well-typed* if some typing judgement can be inferred for it.

The type-inference rules, given in Figure 1, are expressed in a natural-deduction style. Each rule is to be applied in the context of a set $\mathcal{A}$ of assumptions about the types of the free variables appearing in the terms, where each assumption in $\mathcal{A}$ has the form $X : I/J \Rightarrow O$. Rules other than the recursion rule are applicable if under assumptions $\mathcal{A}$ the judgements in the premises can be inferred, and in that case the judgement in the conclusion can also be inferred under the same assumptions $\mathcal{A}$. The rule for recursion is slightly different, in that in order to establish the premise one is permitted to add to the set $\mathcal{A}$ an additional assumption about the recursive variable X. This additional assumption is discharged by the rule, so that the conclusion is inferable under assumptions $\mathcal{A}$ without the additional assumption on X. Since the set $\mathcal{A}$ is the same in the premises and conclusion of each rule except the rule for recursion, to avoid clutter, we have not explicitly indicated the set $\mathcal{A}$ in each case.

In the sequel, we will use the notation $\mathcal{A} \vdash t : \phi$ to assert that there is an inference of the judgement $t : \phi$ from the set of hypotheses $\mathcal{A}$. We will use $\vdash t : \phi$ to assert that a typing judgement $t : \phi$ is inferable from the empty set of assumptions. Note that this is only possible if t is closed.

It is worth pointing out that the type-inference rules do not uniquely associate a type with each well-typed term. The simplest case of this is the rule for nil, which permits any judgment of the form $\text{nil} : \emptyset \Rightarrow O$ to be inferred. However, as we will show later, if a closed term t is well-typed, then in fact there is a uniquely determined set I and a *smallest* set O such that a judgment $t : I \Rightarrow O$ is inferable.

$$\frac{t : I \Rightarrow O \quad a \in I}{a_{(w)}\,?\,t : \{a\}/I \Rightarrow O} \qquad \frac{t : I \Rightarrow O \quad b \notin I}{b_{(r)}\,!\,t : \emptyset/I \Rightarrow O \cup \{b\}} \qquad \frac{t : I \Rightarrow O}{\tau_{(r)} \cdot t : \emptyset/I \Rightarrow O}$$

$$\frac{t_1 : I_1/J \Rightarrow O_1 \quad t_2 : I_2/J \Rightarrow O_2}{t_1 + t_2 : I_1 \cup I_2/J \Rightarrow O_1 \cup O_2} \qquad \frac{t_1 : I_1 \Rightarrow O_1' \quad t_2 : I_2 \Rightarrow O_2' \quad O_1' \subseteq O_1 \quad O_2' \subseteq O_2}{t_1 \;{}_{O_1}\|_{O_2}\; t_2 : (I_1 \cup I_2)\backslash(O_1 \cup O_2) \Rightarrow O_1 \cup O_2}$$

$$\frac{t : I \Rightarrow O \quad a \in I \quad a' \notin I \cup O}{t\{a \leftarrow a'\} : (I\backslash\{a\}) \cup \{a'\} \Rightarrow O} \qquad \frac{t : I \Rightarrow O \quad b' \notin I \cup O}{t\{b \leftarrow b'\} : I \Rightarrow (O\backslash\{b\}) \cup \{b'\}}$$

$$\frac{t : I \Rightarrow O' \quad O \subseteq O'}{t\,[O] : I \Rightarrow O} \qquad \text{nil} : \emptyset \Rightarrow O \qquad \frac{X : I \Rightarrow O \;\vdash\; t : I \Rightarrow O}{\mu X.t : I \Rightarrow O}$$

Fig. 1. Type-Inference Rules

4.2 Transitions

The transition rules for the PIOA language are used to infer transitions of one of the following three types: $t \xrightarrow[w]{a?} u$, $t \xrightarrow[r]{b!} u$, or $t \xrightarrow[r]{\tau} u$. The first of these denotes an *input transition* having associated action a and *weight* w. The second denotes an *output transition* having associated action b and *rate* r. The third denotes an *internal transition* having associated rate r. Both weights w and rates r are required to be positive real numbers, however we regard weights w as dimensionless quantities (such as probabilities) and we regard rates as dimensional quantities with units of 1/time. The full set of transition rules is given in Figure 2.

There are several points to be noted about the transition rules. In the rules for a parallel composition $t_1 \;{}_{O_1}\|_{O_2}\; t_2$, an input transition for component t_1 can occur either independently, if the associated action a is in neither the input set I_2 of t_2 nor the set O_2 of outputs declared to be controlled by t_2, or as a synchronized input transition, if a is in both I_1 and I_2, or else as a synchronized output transition, if a is in I_1 and O_2. Synchronization in a parallel composition results in multiplication of the values that label the transitions. However, note that the rules only call for the multiplication of two weights, or the multiplication of a weight and a rate, but never the multiplication of two rates. This is consistent with our view of weights as dimensionless quantities (*e.g.* probabilities) and with rates as quantities with dimensions of 1/time.

In a parallel composition $t_1 \;{}_{O_1}\|_{O_2}\; t_2$ the syntax declares explicitly the sets O_1 and O_2 of outputs that are to be controlled by t_1 and t_2, respectively. The sets of outputs O_1' and O_2' that t_1 and t_2 can actually produce may be smaller. The reason for this is because as t_1 and t_2 evolve, the sets of outputs that they are capable of actually producing may diminish, though in a parallel composition they still exert control over "lost" output actions by inhibiting their occurrence as inputs in other components.

$$a_{(w)}\,?\,t \xrightarrow[w]{a?} t \qquad b_{(r)}\,!\,t \xrightarrow[r]{b!} t \qquad \tau_{(r)} \cdot t \xrightarrow[r]{\tau} t$$

$$\frac{t_1 \xrightarrow[w]{a?} t'}{t_1 + t_2 \xrightarrow[w]{a?} t'} \qquad \frac{t_2 \xrightarrow[w]{a?} t'}{t_1 + t_2 \xrightarrow[w]{a?} t'} \qquad \frac{t_1 \xrightarrow[r]{b!} t'}{t_1 + t_2 \xrightarrow[r]{b!} t'} \qquad \frac{t_2 \xrightarrow[r]{b!} t'}{t_1 + t_2 \xrightarrow[r]{b!} t'}$$

$$\frac{t_1 \xrightarrow[r]{\tau} t'}{t_1 + t_2 \xrightarrow[r]{\tau} t'} \qquad \frac{t_2 \xrightarrow[r]{\tau} t'}{t_1 + t_2 \xrightarrow[r]{\tau} t'}$$

$$\frac{t_1 \xrightarrow[w]{a?} t_1' \quad t_2 : I_2 \Rightarrow O_2' \quad a \notin I_2 \cup O_2}{t_1 \, {}_{O_1}\|_{O_2} \, t_2 \xrightarrow[w]{a?} t_1' \, {}_{O_1}\|_{O_2} \, t_2} \qquad \frac{t_1 : I_1 \Rightarrow O_1' \quad a \notin I_1 \cup O_1 \quad t_2 \xrightarrow[w]{a?} t_2'}{t_1 \, {}_{O_1}\|_{O_2} \, t_2 \xrightarrow[w]{a?} t_1 \, {}_{O_1}\|_{O_2} \, t_2'}$$

$$\frac{t_1 \xrightarrow[w_1]{a?} t_1' \quad t_2 \xrightarrow[w_2]{a?} t_2'}{t_1 \, {}_{O_1}\|_{O_2} \, t_2 \xrightarrow[w_1 w_2]{a?} t_1' \, {}_{O_1}\|_{O_2} \, t_2'}$$

$$\frac{t_1 \xrightarrow[r]{b!} t_1' \quad t_2 : I_2 \Rightarrow O_2' \quad b \notin I_2}{t_1 \, {}_{O_1}\|_{O_2} \, t_2 \xrightarrow[r]{b!} t_1' \, {}_{O_1}\|_{O_2} \, t_2} \qquad \frac{t_1 : I_1 \Rightarrow O_1' \quad b \notin I_1 \quad t_2 \xrightarrow[r]{b!} t_2'}{t_1 \, {}_{O_1}\|_{O_2} \, t_2 \xrightarrow[r]{b!} t_1 \, {}_{O_1}\|_{O_2} \, t_2'}$$

$$\frac{t_1 \xrightarrow[r]{a!} t_1' \quad t_2 \xrightarrow[w]{a?} t_2'}{t_1 \, {}_{O_1}\|_{O_2} \, t_2 \xrightarrow[wr]{a!} t_1' \, {}_{O_1}\|_{O_2} \, t_2'} \qquad \frac{t_1 \xrightarrow[w]{a?} t_1' \quad t_2 \xrightarrow[r]{a!} t_2'}{t_1 \, {}_{O_1}\|_{O_2} \, t_2 \xrightarrow[wr]{a!} t_1' \, {}_{O_1}\|_{O_2} \, t_2'}$$

$$\frac{t_1 \xrightarrow[r]{\tau} t_1'}{t_1 \, {}_{O_1}\|_{O_2} \, t_2 \xrightarrow[r]{\tau} t_1' \, {}_{O_1}\|_{O_2} \, t_2} \qquad \frac{t_2 \xrightarrow[r]{\tau} t_2'}{t_1 \, {}_{O_1}\|_{O_2} \, t_2 \xrightarrow[r]{\tau} t_1 \, {}_{O_1}\|_{O_2} \, t_2'}$$

$$\frac{t \xrightarrow[w]{a?} t'}{t\;[O] \xrightarrow[w]{a?} t'\;[O]} \qquad \frac{t \xrightarrow[r]{b!} t' \quad b \in O}{t\;[O] \xrightarrow[r]{b!} t'\;[O]} \qquad \frac{t \xrightarrow[r]{b!} t' \quad b \notin O}{t\;[O] \xrightarrow[r]{\tau} t'\;[O]} \qquad \frac{t \xrightarrow[r]{\tau} t'}{t\;[O] \xrightarrow[r]{\tau} t'\;[O]}$$

$$\frac{t \xrightarrow[w]{a?} t'}{t\{a \leftarrow a'\} \xrightarrow[w]{a'?} t'\{a \leftarrow a'\}} \qquad \frac{t \xrightarrow[w]{a?} t' \quad a \neq e}{t\{e \leftarrow e'\} \xrightarrow[w]{a?} t'\{e \leftarrow e'\}}$$

$$\frac{t \xrightarrow[r]{b!} t'}{t\{b \leftarrow b'\} \xrightarrow[r]{b'!} t'\{b \leftarrow b'\}} \qquad \frac{t \xrightarrow[r]{b!} t' \quad b \neq e}{t\{e \leftarrow e'\} \xrightarrow[r]{b!} t'\{e \leftarrow e'\}} \qquad \frac{t \xrightarrow[r]{\tau} t'}{t\{e \leftarrow e'\} \xrightarrow[r]{\tau} t'\{e \leftarrow e'\}}$$

$$\frac{t[\mu X.t/X] \xrightarrow[w]{a?} t'}{\mu X.t \xrightarrow[w]{a?} t'} \qquad \frac{t[\mu X.t/X] \xrightarrow[r]{b!} t'}{\mu X.t \xrightarrow[r]{b!} t'} \qquad \frac{t[\mu X.t/X] \xrightarrow[r]{\tau} t'}{\mu X.t \xrightarrow[r]{\tau} t'}$$

Fig. 2. Transition Rules

5 Metatheory

In this section, we present a number of results targeted at showing that the typing and transition rules presented in the previous section are sensible and interact properly. In particular, we have the following:

- A principal type theorem (Theorem 1).
- A connection between the types that can be inferred for a term and the transitions that can be inferred for it. (Theorems 2 and 3).
- A subject reduction theorem (Theorem 4): well-typedness is preserved across inferable transitions.

5.1 Principal Types

Our first result in this section states that inferable types have disjoint sets of inputs and outputs, and that the set of inputs available on the first transition is contained in the set of inputs available on subsequent transitions.

Lemma 1. *Suppose* $\vdash t : I/J \Rightarrow O$*. Then* $I \subseteq J$ *and* $J \cap O = \emptyset$*.*

It is tempting to think that if $\vdash t : I/J \Rightarrow O$, then $\vdash t : I/J \Rightarrow O'$ for all $O' \supseteq O$ such that $J \cap O' = \emptyset$. However this result does *not* hold for our type system. As a trivial example, if t is the term "nil $[\emptyset]$", then although $\vdash t : \emptyset \Rightarrow \emptyset$, we do not have $\vdash t : \emptyset \Rightarrow O$ for any nonempty O.

Theorem 1 (Principal Type Theorem). *If* $\vdash t : I/J \Rightarrow O$ *for some* I*,* J*, and* O*, then there exists* $\hat{O}$ *such that* $\vdash t : I/J \Rightarrow \hat{O}$*, and such that whenever* $\vdash t : I'/J' \Rightarrow O'$ *then* $I' = I$*,* $J' = J$*, and* $O' \supseteq \hat{O}$*.*

For a given closed, well-typed term t, define the *principal type* of t to be the type $I/J \Rightarrow \hat{O}$ given by Theorem 1. Let $\mathrm{Proc}_{I,O}$ denote the set of all well-typed closed terms t having principal type $I \Rightarrow O'$ for some $O' \subseteq O$.

5.2 Types and Transitions

We next establish connections between the types inferable for a term and the transitions inferable for that term. In particular, if a judgement $t : I/J \Rightarrow O$ is inferable, then I is precisely the set of actions a for which a transition of the form $t \xrightarrow[w]{a?} t'$ is inferable, and O contains all actions b for which a transition of the form $t \xrightarrow[r]{b!} t'$ is inferable. Moreover, well-typedness is preserved across transitions, although inferable types are not preserved exactly due to the possibility that the capacity of producing a particular output action can be lost as a result of taking a transition.

A term t such that $t : I/J \Rightarrow O$ is called *input-enabled* if for all actions $e \in I$ some transition of the form $t \xrightarrow[w]{e?} t'$ is inferable.

Theorem 2 (Input Enabledness Theorem). *Suppose $t : I/J \Rrightarrow O$. Then for all actions e, $e \in I$ if and only if a transition of the form $t \xrightarrow[w]{e?} t'$ is inferable.*

Theorem 3. *Suppose $t : I/J \Rrightarrow O$. Then for all actions e, if a transition of the form $t \xrightarrow[r]{e!} t'$ is inferable, then $e \in O$.*

Theorem 4 (Subject Reduction Theorem). *Suppose $\vdash t : I/J \Rrightarrow O$. If for some term t' a transition of the form $t \xrightarrow[w]{e?} t'$, $t \xrightarrow[r]{e!} t'$, or $t \xrightarrow[r]{\tau} t'$ is inferable, then $\vdash t' : J/J \Rrightarrow O'$ for some $O' \subseteq O$. In particular,* $\mathrm{Proc}_{I,O}$ *is closed under transitions.*

5.3 Total Transition Weight/Rate

For given terms t and t' and action e, the transition inference rules may yield zero or more distinct inferences of transitions of one of the forms: $t \xrightarrow[w]{e?} t'$, $t \xrightarrow[r]{e!} t'$, or $t \xrightarrow[r]{\tau} t'$, where w and r vary depending on the specific inference. However, it is a consequence of the requirement that all recursive variables be guarded by a prefixing operation that there can be only finitely many such inferences. We write $t \overset{e?}{\underset{w}{\longmapsto}} t'$ to assert that w is the sum of all the weights w_i appearing in distinct inferences of transitions of the form $t \xrightarrow[w_i]{e?} t'$. We call such an expression a *total transition*. Since there are only finitely many such inferences, the sum w is finite. In case there are no inferable transitions $t \xrightarrow[w_i]{e?} t'$ we write $t \overset{e?}{\underset{0}{\longmapsto}} t'$. For output and internal transitions, the notations $t \overset{e!}{\underset{r}{\longmapsto}} t'$ and $t \overset{\tau}{\underset{r}{\longmapsto}} t'$ are defined similarly.

A related notation will also be useful. Suppose t and t' are closed terms in $\mathrm{Proc}_{I,O}$. Then for all $e \in \mathit{Act} \cup \{\tau\}$ define $\Delta_e^O(t,t')$ as follows:

1. If $e \in I$, then $\Delta_e^O(t,t')$ is the unique weight w for which $t \overset{e?}{\underset{w}{\longmapsto}} t'$.
2. If $e \in O$, then $\Delta_e^O(t,t')$ is the unique rate r for which $t \overset{e!}{\underset{r}{\longmapsto}} t'$.
3. If $e = \tau$, then $\Delta_e^O(t,t')$ is the unique rate r for which $t \overset{\tau}{\underset{r}{\longmapsto}} t'$.
4. If $e \notin I \cup O \cup \{\tau\}$ then $\Delta_e^O(t,t) = 1$ and $\Delta_e^O(t,t') = 0$ if $t' \neq t$.

The *derivative* of term t by action e is the mapping $\Delta_e^O t : \mathrm{Proc}_{I,O} \to [0,\infty)$ defined so that the relation $(\Delta_e^O t)(t') = \Delta_e^O(t,t')$ holds identically for all terms t'. If S is a set of terms, then we use $\Delta_e^O(t,S)$ or $(\Delta_e^O t)(S)$ to denote the sum $\sum_{t' \in S} \Delta_e^O(t,t')$, which is finite.

Note that the reason why we retain the superscripted O in the Δ_e^O notation is because the terms t and t' do not uniquely determine the set O, therefore whether clause (2) or (4) in the definition applies for a given action e depends on the set O.

Define the class of *input-stochastic* terms to be the largest subset of $\mathrm{Proc}_{I,O}$ such that if t is input-stochastic then the following conditions hold:

1. For all $e \in I$ we have $\sum_{t'} \Delta_e^O(t, t') = 1$.
2. Whenever $\Delta_e^O(t, t') > 0$ then t' is also input-stochastic.

Input-stochastic terms are those for which the weights associated with input transitions can be interpreted as probabilities. These are the terms that are naturally associated with PIOA, in the sense that the set of all stochastic terms in $\mathrm{Proc}_{I,O}$ is the set of states of a PIOA with input actions I, output actions O, and the single internal action τ, and with Δ_e^O as the "transition matrix" for action e.

In a later section, we will require the notion of the *total rate* $\mathrm{rt}(t)$ of a closed, well-typed term t such that $\vdash t : I \Rightarrow O$. This quantity is defined as follows:

$$\mathrm{rt}(t) = \sum_{e \in O \cup \{\tau\}} \sum_{t'} \Delta_e^O(t, t').$$

It is a consequence of the fact that only finitely many actions e can appear in term t that $\mathrm{rt}(t)$ is finite. Note also that $\mathrm{rt}(t)$ does not depend on O.

6 Equivalence of Terms

In this section, we define two notions of equivalence for our language, and investigate their properties. The first equivalence, which we call *weighted bisimulation equivalence*, is a variant of bisimulation that is based on the same ideas as *probabilistic bisimulation* [LS91], Hillston's "strong equivalence" [Hil96], and "strong Markovian bisimulation" [HH02]. The second equivalence, called *PIOA behavior equivalence*, is based on the notion of the "behavior map" associated with a PIOA, which has appeared in various forms in our previous work [WSS94,WSS97,SS98], along with motivation for the concept. Additional motivation and a detailed comparison of probabilistic bisimulation equivalence and PIOA behavior equivalence can be found in [Sta03]. In the present paper we focus primarily on congruence properties of these equivalences with respect to the operators of our language.

6.1 Weighted Bisimulation Equivalence

A *weighted bisimulation* is an equivalence relation R on $\mathrm{Proc}_{I,O}$ such that whenever $t \ R \ t'$ then for all actions e and all equivalence classes $\mathcal{C}$ of R we have $\Delta_e^O(t, \mathcal{C}) = \Delta_e^O(t', \mathcal{C})$. Clearly, the identity relation is a weighted bisimulation. It is a standard argument to prove that the transitive closure of the union of an arbitrary collection of weighted bisimulations is again a weighted bisimulation. Thus, there exists a largest weighted bisimulation $\underset{I,O}{\sim}$ on $\mathrm{Proc}_{I,O}$. We call $\underset{I,O}{\sim}$ the *weighted bisimulation equivalence* relation.

Define a *weighting* on terms to be a function μ from $\mathrm{Proc}_{I,O}$ to the non-negative real numbers, such that that $\mu(t) = 0$ for all but finitely many terms $t \in \mathrm{Proc}_{I,O}$. Suppose R is an equivalence relation on $\mathrm{Proc}_{I,O}$. Define the *lifting* of R to weightings to be the relation $\overline{R}$ on weightings defined by the following condition: $\mu \ \overline{R} \ \mu'$ if and only if $\mu(\mathcal{C}) = \mu'(\mathcal{C})$ for all equivalence classes $\mathcal{C}$ of R.

The following result (*cf.* [JLY01]) simply restates the definition of weighted bisimulation in terms of weightings.

Lemma 2. *An equivalence relation R on $\mathrm{Proc}_{I,O}$ is a weighted bisimulation if and only if $t\ R\ u$ implies $\Delta^O_e t\ \overline{R}\ \Delta^O_e u$ for all terms t, u and all actions e.*

Lemma 3. *Let R be a symmetric relation on terms. If for all terms t, u and all actions e we have*

$$t\ R\ u \text{ implies } \Delta^O_e t\ \overline{(R \cup \underset{I,O}{\sim})^*}\ \Delta^O_e u,$$

then $R \subseteq \underset{I,O}{\sim}$.

Lemma 3 can be used to establish that weighted bisimilarity is substitutive with respect to the operators of our language.

Theorem 5. *The following hold, whenever the sets of inputs and outputs are such that the terms are well-typed and the indicated relations make sense:*

1. *If $t \underset{I',O'}{\sim} t'$, then*
   (a) $a_{(w)}\,?\,t \underset{I,O}{\sim} a_{(w)}\,?\,t'$
   (b) $b_{(r)}\,!\,t \underset{I,O}{\sim} b_{(r)}\,!\,t'$
   (c) $\tau_{(r)} \cdot t \underset{I,O}{\sim} \tau_{(r)} \cdot t'$
2. *If $t_1 \underset{I_1,O_1}{\sim} t'_1$ and $t_2 \underset{I_2,O_2}{\sim} t'_2$, then $t_1 + t_2 \underset{I,O}{\sim} t'_1 + t'_2$*
3. *If $t_1 \underset{I_1,O_1}{\sim} t'_1$ and $t_2 \underset{I_2,O_2}{\sim} t'_2$, then $t_1 \,{}_{O_1}\|_{O_2}\, t_2 \underset{I,O}{\sim} t'_1 \,{}_{O_1}\|_{O_2}\, t'_2$.*
4. *If $t \underset{I',O'}{\sim} t'$, then $t\ [O] \underset{I,O}{\sim} t'\ [O]$.*
5. *If $t \underset{I',O'}{\sim} t'$, then $t\{e \leftarrow e'\} \underset{I,O}{\sim} t'\{e \leftarrow e'\}$.*

6.2 Behavior Equivalence

In this section, we restrict our attention to the fragment of the language obtained by omitting internal actions and hiding. Let $\mathrm{Proc}^-_{I,O}$ denote the portion of $\mathrm{Proc}_{I,O}$ contained in this fragment. The full language can be treated, but the definition of behavior equivalence becomes more complicated and requires the use of fixed-point techniques, rather than the simple inductive definition given below.

Behavior equivalence is defined by associating with each closed term t with $\vdash t : I \Rightarrow O$ a certain function $\mathcal{B}^O_t$ which we call the *behavior* of t. Terms t and t' will be called *behavior equivalent* if their associated behaviors are identical.

To define $\mathcal{B}^O_t$, some preliminary definitions are required. A *rated action* is a pair $(r, e) \in [0, \infty) \times Act$. Rather than the somewhat heavy notation (r, e), we usually denote a rated action by an expression ${}_re$ in which the rate appears as a subscript preceding the action. A finite sequence

$${}_{r_1}e_1\,{}_{r_2}e_2 \cdots {}_{r_n}e_n$$

of rated actions is called a *rated trace*. We use ϵ to denote the empty rated trace.

An *observable* is a mapping from rated traces to real numbers. We use Obs to denote the set of all observables. The *derivative* of an observable Φ by a rated action ${}_re$ is the observable Ψ defined by $\Psi(\alpha) = \Phi({}_re\ \alpha)$ for all rated traces α. Borrowing notation from the literature on formal power series (of which observables are an example), we write ${}_re^{-1}\Phi$ to denote the derivative of Φ by the rated action ${}_re$.

To each term t in $\mathrm{Proc}^-_{I,O}$ we associate a *transformation of observables*:

$$\mathcal{B}^O_t : \mathrm{Obs} \to \mathrm{Obs}$$

according to the following inductive definition:

$$\mathcal{B}^O_t[\Phi](\epsilon) = \Phi(\epsilon)$$

$$\mathcal{B}^O_t[\Phi]({}_re\ \alpha) = \sum_{t'} \Delta^O_e(t,t')\ \mathcal{B}^O_{t'}[{}_{r+\mathrm{rt}(t)}e^{-1}\Phi](\alpha).$$

Terms t and t' in $\mathrm{Proc}^-_{I,O}$ are called *behavior equivalent*, and we write $t \underset{I,O}{\equiv} t'$, if $\mathcal{B}^O_t = \mathcal{B}^O_{t'}$.

Intuitively, in the definition of $\mathcal{B}^O_t[\Phi](\alpha)$, one should think of the rated trace α as giving certain partial information about a particular set of execution trajectories that might be traversed by a process t in combination with its environment. In particular, if $\alpha = {}_{r_1}e_1{}_{r_2}e_2 \ldots {}_{r_n}e_n$, then $e_1e_2 \ldots e_n$ is the sequence of actions performed in such a trajectory (including both input and output actions) and $r_1r_2 \ldots r_n$ is the sequence of output rates associated with the successive states visited by the environment in such a trajectory. The observable Φ should be thought of as a way of associating some numeric measure, or reward, with trajectories. By "unwinding" the definition of $\mathcal{B}^O_t[\Phi](\alpha)$, one can see that it amounts to a weighted summation of the rewards $\Phi(\alpha')$ associated with trajectories α' that start from t and that "match" α, in the sense that α and α' have the same sequence of actions, but the rates of actions in α' are obtained by adding to the rate of the corresponding action in α the total rate $\mathrm{rt}(u)$ of a state u reachable by process t. Further explanation and examples of what can be done with behavior maps can be found in [SS98,Sta03].

The next result states that behavior equivalent terms have the same total rate, and the same total transition weight for each individual action.

Lemma 4. *Suppose $t \underset{I,O}{\equiv} t'$. Then*

1. $\sum_u \Delta^O_e(t,u) = \sum_u \Delta^O_e(t',u)$ *for all* $e \in Act$.
2. $\mathrm{rt}(t) = \mathrm{rt}(t')$.

A mistake that we made repeatedly while developing the language and these results was to suppose that the choice operator in the language ought to correspond to sum of behavior maps. This is wrong. The following result shows the correct relationship.

Lemma 5. *Suppose t_1 and t_2 are terms, such that $\vdash t_1 + t_2 : I \Rightarrow O$. Then for all observables Φ, rated actions ${}_r e$, and rated traces α':*

$$\mathcal{B}^O_{t_1+t_2}[\Phi](\epsilon) = \Phi(\epsilon)$$
$$\mathcal{B}^O_{t_1+t_2}[\Phi]({}_r e\ \alpha') = \mathcal{B}^O_{t_1}[\Phi]({}_{r+\mathrm{rt}(t_2)} e\ \alpha') + \mathcal{B}^O_{t_2}[\Phi]({}_{r+\mathrm{rt}(t_1)} e\ \alpha').$$

The following result states that behavior maps are compositional with respect to the parallel operator. We have proved this result in various forms in our previous papers [WSS94,WSS97,SS98]. A proof of the result based on the specific definition of behavior map given here appears in [Sta03].

Lemma 6. *Suppose t_1 and t_2 are terms, such that $\vdash t_1\ {}_{O_1}\|_{O_2}\ t_2 : I \Rightarrow O$. Then*

$$\mathcal{B}^O_{t_1\ {}_{O_1}\|_{O_2}\ t_2} = \mathcal{B}^{O_1}_{t_1} \circ \mathcal{B}^{O_2}_{t_2}.$$

Lemma 7. *Suppose $\vdash t\{e \leftarrow e'\} : I \Rightarrow O$. Let mapping h on rated traces be the string homomorphism that interchanges ${}_r e'$ and ${}_r e$ and is the identity mapping on all other rated actions. Then*

$$\mathcal{B}^O_{t\{e \leftarrow e'\}}[\Phi] = \mathcal{B}^O_t[\Phi \circ h] \circ h,$$

where $O' = O$ if $e' \in I$, and $O' = (O \setminus \{e'\}) \cup \{e\}$ if $e' \in O$.

The preceding lemmas can be used to show that behavior equivalence is substitutive with respect to the operations of our language (exclusive of internal prefixing and hiding). The proofs are all ultimately by induction on the length of the rated trace α supplied as argument, though in the cases of parallel composition and renaming we have been able to hide this "operational" induction inside the more "denotational" Lemmas 6 and 7.

Theorem 6. *The following hold, whenever the sets of inputs and outputs are such that the terms are well-typed and the indicated relations make sense:*

1. *If $t \underset{I',O'}{\equiv} t'$, then*
   (a) *$a_{(w)}\,?\,t \underset{I,O}{\equiv} a_{(w)}\,?\,t'$*
   (b) *$b_{(r)}\,!\,t \underset{I,O}{\equiv} b_{(r)}\,!\,t'$*
2. *If $t_1 \underset{I_1,O_1}{\equiv} t_1'$ and $t_2 \underset{I_2,O_2}{\equiv} t_2'$, then $t_1 + t_2 \underset{I,O}{\equiv} t_1' + t_2'$*
3. *If $t_1 \underset{I_1,O_1}{\equiv} t_1'$ and $t_2 \underset{I_2,O_2}{\equiv} t_2'$, then $t_1\ {}_{O_1}\|_{O_2}\ t_2 \underset{I,O}{\equiv} t_1'\ {}_{O_1}\|_{O_2}\ t_2'$.*
4. *If $t \underset{I',O'}{\equiv} t'$, then $t\{e \leftarrow e'\} \underset{I,O}{\equiv} t'\{e \leftarrow e'\}$.*

6.3 Comparison of the Equivalences

The following result is a consequence of characterizations, obtained in [Sta03], of weighted bisimulation equivalence and behavior equivalence.

Theorem 7. *Suppose t and t' are in $\mathrm{Proc}^-_{I,O}$. If $t \underset{I,O}{\sim} t'$, then $t \underset{I,O}{\equiv} t'$.*

In addition, if $I = \emptyset$ and $O = \{a, b, c\}$ then we have

$$a_{(1)}\,!\,b_{(2)}\,!\,\mathrm{nil} + a_{(1)}\,!\,c_{(2)}\,!\,\mathrm{nil} \underset{I,O}{\equiv} a_{(2)}\,!\,(b_{(1)}\,!\,\mathrm{nil} + c_{(1)}\,!\,\mathrm{nil}),$$

but the same two terms are not related by $\underset{I,O}{\sim}$. Thus, weighted bisimulation equivalence is a strict refinement of behavior equivalence.

7 Conclusion

We have presented a process-algebraic language having input, output, and internal transitions, where input actions are labeled by weights and output and internal actions are labeled by rates. A set of typing rules is employed to define the sets $\mathrm{Proc}_{I,O}$ of well-typed terms, which are guaranteed to have transitions enabled for all actions $a \in I$. A readily identifiable subset of the well-typed terms are the input-stochastic terms, in which input weights can be interpreted as probabilities. The input-stochastic terms are therefore the states of a PIOA, so that the language is suitable for writing PIOA-based specifications. We have defined two equivalences on the language, a weighted bisimulation equivalence defined in the same pattern as the classical probabilistic bisimulation equivalence, and a so-called "behavior equivalence" whose definition is motivated by our previous work on PIOA. Both equivalences were shown to be congruences, and we noted that weighted bisimulation equivalence is a strict refinement of behavior equivalence.

A natural direction for future work is to axiomatize the equational theories of the two congruences. For weighted bisimulation equivalence, a standard equational axiomatization should be possible, and is not likely to yield any surprises. The situation for behavior equivalence is a bit different, however. Weighted bisimulation equivalence is the largest equivalence on terms that respects transition weights in the sense of Lemma 2. Since behavior equivalence relates terms that are not weighted bisimulation equivalent, it will not be possible to obtain an equational axiomatization of behavior equivalence, at least in the context of a theory of equations between terms. However, it appears that it is possible to obtain an axiomatization of behavior equivalence in the context of a theory of equations between *weightings*, rather than terms. We are currently working out the details of this idea.

References

[BDG98] M. Bernardo, L. Donatiello, and R. Gorrieri. A formal approach to the integration of performance aspects in the modeling and analysis of concurrent systems. *Information and Computation*, 144(2):83–154, 1998.

[BH01] E. Brinksma and H. Hermanns. Process algebra and Markov chains. In E. Brinksma, H. Hermanns, and J.-P. Katoen, editors, *FMPA 2000: Euro-Summerschool on Formal Methods and Performance Analysis*, volume 2090 of *Lecture Notes in Computer Science*, pages 183–231. Springer-Verlag, 2001.

[Buc99] P. Buchholz. Exact performance equivalence: An equivalence relation for stochastic automata. *Theoretical Computer Science*, 215:263–287, 1999.

[CPS93] R. Cleaveland, J. Parrow, and B. U. Steffen. The Concurrency Workbench: A semantics-based tool for the verification of concurrent systems. *ACM TOPLAS*, 15(1), 1993.

[DNS95] R. De Nicola and R. Segala. A process algebraic view of Input/Output Automata. *Theoretical Computer Science*, 138(2), 1995.

[GL00] S. J. Garland and N. A. Lynch. Using I/O automata for developing distributed systems. In Gary T. Leavens and Murali Sitaraman, editors, *Foundations of Component-Based Systems*, pages 285–312. Cambridge University Press, 2000.

[HH02] J.-P. Katoen H. Hermanns, U. Herzog. Process algebra for performance evaluation. *Theoretical Computer Science*, 274:43–97, 2002.

[Hil94] J. Hillston. The nature of synchronization. In U. Herzog and M. Rettelbach, editors, *Proceedings of the 2nd Workshop on Process Algebra and Performance Modeling*, pages 51–70, University of Erlangen, July 1994.

[Hil96] J. Hillston. *A Compositional Approach to Performance Modelling.* Cambridge University Press, 1996.

[JLY01] B. Jonsson, K. G. Larsen, and W. Yi. Probabilistic extensions of process algebras. In J.A. Bergstra, A. Ponse, and S.A. Smolka, editors, *Handbook of Process Algebra.* Elsevier, 2001.

[LS91] K. G. Larsen and A. Skou. Bisimulation through probabilistic testing. *Information and Computation*, 94(1):1–28, September 1991.

[LT87] N. A. Lynch and M. Tuttle. Hierarchical correctness proofs for distributed algorithms. In *Proceedings of the 6th Annual ACM Symposium on Principles of Distributed Computing*, pages 137–151, 1987.

[PA91] B. Plateau and K. Atif. Stochastic automata networks for modeling parallel systems. *IEEE Transactions on Software Engineering*, 17:1093–1108, 1991.

[SS98] E. W. Stark and S. Smolka. Compositional analysis of expected delays in networks of probabilistic I/O automata. In *Proc. 13th Annual Symposium on Logic in Computer Science*, pages 466–477, Indianapolis, IN, June 1998. IEEE Computer Society Press.

[Sta03] E. Stark. On behavior equivalence for probabilistic I/O automata and its relationship to probabilistic bisimulation. *Journal of Automata, Languages, and Combinatorics*, 8(2), 2003. to appear.

[Vaa91] F. W. Vaandrager. On the relationship between process algebra and input/output automata. In *Sixth Annual Symposium on Logic in Computer Science (LICS '91)*, pages 387–398, Amsterdam, July 1991. Computer Society Press.

[WSS94] S.-H. Wu, S. A. Smolka, and E. W. Stark. Compositionality and full abstraction for probabilistic I/O automata. In *Proceedings of CONCUR '94 — Fifth International Conference on Concurrency Theory*, Uppsala, Sweden, August 1994.

[WSS97] S.-H. Wu, S. A. Smolka, and E. W. Stark. Composition and behaviors of probabilistic I/O automata. *Theoretical Computer Science*, 176(1-2):1–38, 1997.

[ZCS03] D. Zhang, R. Cleaveland, and E.W. Stark. The integrated CWB-NC/PIOATool for functional and performance analysis of concurrent systems. In *Proceedings of the Ninth International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS 2003).* Lecture Notes in Computer Science, Springer-Verlag, April 2003.