Logical Properties of Name Restriction

Luca Cardelli, Andrew D. Gordon

Microsoft Research

DRAFT

Abstract. We extend the modal logic of ambients described in [7] to the full ambient calculus, including name restriction. We introduce logical operators that can be used to make assertions about restricted names, and we study their properties.

1 Introduction

The π-calculus notion of name restriction [11], initially intended to represent hidden communication channels, has been used also to represent hidden encryption keys [2] and as the basis for definitions of secrecy [2, 4]. In the context of the Ambient Calculus [6], name restriction can be used to represent hidden locations and (by extrapolating [4] and [5]) secret locations. In general, we would like to have process calculi where we can represent protocols for creating shared encryption keys and secret locations; name restriction seems crucial to all this.

In π-calculus notation, $(\nu n)P$ is a restriction of the name n in the process P, meaning that n is not currently known outside the scope of P. The prefix (νn) is more a bookkeeping device than a barrier. It is quite possible for P to communicate n to some external process; then the restriction (νn) must be formally pushed outwards to encompass the new scope of n and maintain the scoping invariant; this procedure is called *name extrusion*. Processes are considered equivalent up to extrusion; that is, extrusion is not regarded as a computational step. Conversely, when a name is forgotten in part of a process, the scope of (νn) may be restricted; this is called name *intrusion*. Manipulation of (νn) prefixes includes, in particular, renaming and swapping of prefixes, so that there is no obvious way of talking about "the first restricted name" or any particular restricted name of a process.

The Ambient Calculus can be regarded essentially as an extension of the π-calculus with dynamic location structures. In [7] we present a modal logic for describing properties of Ambient Calculus processes, with particular emphasis on expressing the structure and evolution of hierarchies of locations. Much of that logic can be applied directly to the π-calculus. However, in [7] we left out name restriction; we now intend to fill that gap in a way that can be applied both to the π-calculus, where names are channels, and to the Ambient Calculus, where names are locations. In both cases, we need to investigate the logical properties of name restriction.

In our existing logic we can describe detailed properties of processes. If we now consider restriction, what does it mean to describe properties of restricted names? We would like to be able to say, for example, "a shared key is established between locations a and b", or "a secret location is created that only a and b can access". In a protocol that establishes such shared secrets, the secrets are typically represented by restricted names. The problem is that there is no obvious way to talk about such restricted names in the specification of the protocol. We might be tempted to use ordinary existential quantification, and say "there exists a name shared between locations a and b". But this is not good enough, because we want that name to be fresh and unknown to other locations or potential attackers.

Therefore, we want a new form of quantification that can be read as "in the process there exists a restricted name which we shall call x, and such that $\mathcal{A}$", where x is a variable that ranges over names, and $\mathcal{A}$ is some property that may involve x. Let us indicate this quantifier as $(\nu x)\mathcal{A}$; this formula is meant to correspond somehow to a process of the form $(\nu n)P$ where x denotes n. However, since (νn) can float, the matching of (νx) to any particular (νn) is not obvious.

This means that the logical rules of our tentative $(\nu x)\mathcal{A}$ quantifier are going to be fairly complex, or at least unfamiliar. We have approached this complexity by splitting $(\nu x)\mathcal{A}$ into two operators; one for quantifying over fresh names, and one for mentioning restricted names. The first operator is the Gabbay-Pitts quantifier, $\text{И}x.\mathcal{A}$, adapted to our context: it quantifies over all names that do not occur free either in the formula $\mathcal{A}$ or in the described process. The second is a binary operator (not a quantifier) called *revelation*, $n®\mathcal{A}$, which means that it is possible to reveal a restricted name as the given name n, and then assert $\mathcal{A}$. (Revelation fails to hold if it would lead to a name clash in the process.)

We investigate the properties of $n®\mathcal{A}$ and $\text{И}x.\mathcal{A}$ separately. We combine them to define $(\nu x)\mathcal{A}$ as $\text{И}x.x®\mathcal{A}$, and then we study the derived properties of $(\nu x)\mathcal{A}$.

2 Summary of the Ambient Logic

In this section, we provide a quick summary of the ambient calculus. Although this summary is technically self-contained, we assume some knowledge of [6]: see that paper for discussion and motivation.

We also summarize the ambient logic studied in [7]. Again, this is self-contained, but knowledge of that paper will help. Two new operators, *revelation* and its adjunct *hiding*, are introduced here, and are discussed in the following sections.

2.1 The Calculus

The syntax of the Ambient Calculus is defined in the following table:

Processes

$P,Q,R ::=$	processes	$M ::=$	capabilities
$(\nu n)P$	restriction	n	name
$\mathbf{0}$	void	*in* M	can enter into M
$P \mid Q$	composition	*out* M	can exit out of M
$!P$	replication	*open* M	can open M
$M[P]$	ambient	ε	null
$M.P$	capability action	$M.M'$	path

$(n).P$	input action
$\langle M \rangle$	output action

The set of free names of a process P, written $fn(P)$ is defined as follows, where the only binders are restriction and the input action.

Free names

$fn((\nu n)P) \triangleq fn(P) - \{n\}$	$fn(n) \triangleq \{n\}$
$fn(\mathbf{0}) \triangleq \emptyset$	$fn(in\ M) \triangleq fn(M)$
$fn(P \mid Q) \triangleq fn(P) \cup fn(Q)$	$fn(out\ M) \triangleq fn(M)$
$fn(!P) \triangleq fn(P)$	$fn(open\ M) \triangleq fn(M)$
$fn(M[P]) \triangleq fn(M) \cup fn(P)$	$fn(\varepsilon) \triangleq \emptyset$
$fn(M.P) \triangleq fn(M) \cup fn(P)$	$fn(M.M') \triangleq fn(M) \cup fn(M')$
$fn((n).P) \triangleq fn(P) - \{n\}$	
$fn(\langle M \rangle) \triangleq fn(M)$	

We write $P\{n{\leftarrow}M\}$ for the substitution of the capability M for each free occurrence of the name n in the process P. Similarly for $M\{n{\leftarrow}M'\}$. We identify processes up to renaming of bound names, that is, we assume the identities $(\nu n)P = (\nu m)P\{n{\leftarrow}m\}$ and $(n).P = (m).P\{n{\leftarrow}m\}$, where, in both equations, $m \notin fn(P)$.

We use some syntactic conventions. We use parentheses for precedence. The process $\mathbf{0}$ is often omitted in the contexts $n[\mathbf{0}]$ and $M.\mathbf{0}$, yielding $n[]$ and M. Composition has the weakest binding power, so that the expression $(\nu n)P \mid Q$ is read $((\nu n)P) \mid Q$, the expression $!P \mid Q$ is read $(!P) \mid Q$, the expression $M.P \mid Q$ is read $(M.P) \mid Q$, and the expression $(n).P \mid Q$ is read $((n).P) \mid Q$.

Structural congruence is a relation between processes used as an aid in the definition of reduction. With respect to [6], the structural rules for replication have been refined.

Structural Congruence

$P \equiv P$	(Struct Refl)
$P \equiv Q \Rightarrow Q \equiv P$	(Struct Symm)
$P \equiv Q, Q \equiv R \Rightarrow P \equiv R$	(Struct Trans)
$P \equiv Q \Rightarrow (\nu n)P \equiv (\nu n)Q$	(Struct Res)
$P \equiv Q \Rightarrow P \mid R \equiv Q \mid R$	(Struct Par)
$P \equiv Q \Rightarrow !P \equiv !Q$	(Struct Repl)
$P \equiv Q \Rightarrow n[P] \equiv n[Q]$	(Struct Amb)
$P \equiv Q \Rightarrow M.P \equiv M.Q$	(Struct Action)
$P \equiv Q \Rightarrow (n).P \equiv (n).Q$	(Struct Input)
$\varepsilon.P \equiv P$	(Struct ε)
$(M.M').P \equiv M.M'.P$	(Struct .)

$(\nu n)(\nu m)P \equiv (\nu m)(\nu n)P$	(Struct Res Res)
$(\nu n)\mathbf{0} \equiv \mathbf{0}$	(Struct Res Zero)
$(\nu n)(P \mid Q) \equiv P \mid (\nu n)Q \quad \text{if } n \notin fn(P)$	(Struct Res Par)
$(\nu n)(m[P]) \equiv m[(\nu n)P] \quad \text{if } n \neq m$	(Struct Res Amb)
$P \mid \mathbf{0} \equiv P$	(Struct Par Zero)
$P \mid Q \equiv Q \mid P$	(Struct Par Comm)
$(P \mid Q) \mid R \equiv P \mid (Q \mid R)$	(Struct Par Assoc)
$!\mathbf{0} \equiv \mathbf{0}$	(Struct Repl Zero)
$!(P \mid Q) \equiv !P \mid !Q$	(Struct Repl Par)
$!P \equiv P \mid !P$	(Struct Repl Copy)
$!P \equiv !!P$	(Struct Repl Repl)

The reduction relation, defined in the following table, describes the dynamic behavior of ambients. In particular, the rules (Red In), (Red Out) and (Red Open) represent mobility, while (Red Comm) represents local communication (see [6] for an extended discussion). For example, the process $a[p[out\ a.\ in\ b.\ \langle m\rangle]] \mid b[open\ p.\ (n).\ n[]]$ represents a packet p that travels out of host a and into host b, where it is opened, and its contents m are read and used to create a new ambient. The process reduces in four steps (illustrating each of the four reduction rules) to the residual process $a[] \mid b[m[]]$.

Reduction

$n[in\ m.\ P \mid Q] \mid m[R] \longrightarrow m[n[P \mid Q] \mid R]$	(Red In)
$m[n[out\ m.\ P \mid Q] \mid R] \longrightarrow n[P \mid Q] \mid m[R]$	(Red Out)
$open\ n.\ P \mid n[Q] \longrightarrow P \mid Q$	(Red Open)
$(n).P \mid \langle M\rangle \longrightarrow P\{n \leftarrow M\}$	(Red Comm)
$P \longrightarrow Q \;\Rightarrow\; (\nu n)P \longrightarrow (\nu n)Q$	(Red Res)
$P \longrightarrow Q \;\Rightarrow\; P \mid R \longrightarrow Q \mid R$	(Red Par)
$P \longrightarrow Q \;\Rightarrow\; n[P] \longrightarrow n[Q]$	(Red Amb)
$P' \equiv P, P \longrightarrow Q, Q \equiv Q' \;\Rightarrow\; P' \longrightarrow Q'$	(Red $\equiv$)
$\longrightarrow^*$	reflexive and transitive closure of $\longrightarrow$

2.2 The Logic

The syntax of logical formulas is summarized below. This is a modal predicate logic with classical negation. As usual, many standard connectives are interdefinable; we take $\mathbf{T}$, $\neg$, $\vee$, $\Diamond$, $\forall$ as primitive, and $\mathbf{F}$, $\Rightarrow$, $\wedge$, $\Box$, $\exists$ as derived.

The meaning of the formulas will be given shortly in terms of a satisfaction relation. Informally, the first three formulas (true, negation, disjunction) give propositional logic. The next five (void, composition and its adjunct, location and its adjunct) describe tree-like structures of locations. Revelation and its adjunct are new to this paper, and are discussed in detail later. The two spatial and temporal modalities make assertions about states that may happen “further away” in space or time respectively. Quantified variables range only over names: these variables may appear in the location and revelation constructs, and their adjuncts.

Logical Formulas

η	a name n or a variable x
$\mathcal{A}, \mathcal{B}, \mathcal{C}$ ::=	
$\mathbf{T}$	true
$\neg\mathcal{A}$	negation
$\mathcal{A} \vee \mathcal{B}$	disjunction
$\mathbf{0}$	inaction
$\mathcal{A} \mid \mathcal{B}$	composition
$\mathcal{A} \triangleright \mathcal{B}$	composition adjunct
$\eta[\mathcal{A}]$	location
$\mathcal{A} @ \eta$	location adjunct
$\eta \circledR \mathcal{A}$	revelation
$\mathcal{A} \oslash \eta$	revelation adjunct
$\Diamond \mathcal{A}$	sometime modality
$\text{✧} \mathcal{A}$	somewhere modality
$\forall x.\mathcal{A}$	universal quantification

Free Names and Free Variables

$fn(n)$	$\triangleq$	$\{n\}$	$fv(n)$	$\triangleq$	$\emptyset$
$fn(x)$	$\triangleq$	$\emptyset$	$fv(x)$	$\triangleq$	$\{x\}$
$fn(\mathbf{T})$	$\triangleq$	$\emptyset$	$fv(\mathbf{T})$	$\triangleq$	$\emptyset$
$fn(\neg\mathcal{A})$	$\triangleq$	$fn(\mathcal{A})$	$fv(\neg\mathcal{A})$	$\triangleq$	$fv(\mathcal{A})$
$fn(\mathcal{A} \vee \mathcal{B})$	$\triangleq$	$fn(\mathcal{A}) \cup fn(\mathcal{B})$	$fv(\mathcal{A} \vee \mathcal{B})$	$\triangleq$	$fv(\mathcal{A}) \cup fv(\mathcal{B})$
$fn(\mathbf{0})$	$\triangleq$	$\emptyset$	$fv(\mathbf{0})$	$\triangleq$	$\emptyset$
$fn(\mathcal{A} \mid \mathcal{B})$	$\triangleq$	$fn(\mathcal{A}) \cup fn(\mathcal{B})$	$fv(\mathcal{A} \mid \mathcal{B})$	$\triangleq$	$fv(\mathcal{A}) \cup fv(\mathcal{B})$
$fn(\mathcal{A} \triangleright \mathcal{B})$	$\triangleq$	$fn(\mathcal{A}) \cup fn(\mathcal{B})$	$fv(\mathcal{A} \triangleright \mathcal{B})$	$\triangleq$	$fv(\mathcal{A}) \cup fv(\mathcal{B})$
$fn(\eta[\mathcal{A}])$	$\triangleq$	$fn(\eta) \cup fn(\mathcal{A})$	$fv(\eta[\mathcal{A}])$	$\triangleq$	$fv(\eta) \cup fv(\mathcal{A})$
$fn(\mathcal{A} @ \eta)$	$\triangleq$	$fn(\mathcal{A}) \cup fn(\eta)$	$fv(\mathcal{A} @ \eta)$	$\triangleq$	$fv(\mathcal{A}) \cup fv(\eta)$
$fn(\eta \circledR \mathcal{A})$	$\triangleq$	$fn(\eta) \cup fn(\mathcal{A})$	$fv(\eta \circledR \mathcal{A})$	$\triangleq$	$fv(\eta) \cup fv(\mathcal{A})$
$fn(\mathcal{A} \oslash \eta)$	$\triangleq$	$fn(\mathcal{A}) \cup fn(\eta)$	$fv(\mathcal{A} \oslash \eta)$	$\triangleq$	$fv(\mathcal{A}) \cup fv(\eta)$
$fn(\Diamond \mathcal{A})$	$\triangleq$	$fn(\mathcal{A})$	$fv(\Diamond \mathcal{A})$	$\triangleq$	$fv(\mathcal{A})$
$fn(\text{✧} \mathcal{A})$	$\triangleq$	$fn(\mathcal{A})$	$fv(\text{✧} \mathcal{A})$	$\triangleq$	$fv(\mathcal{A})$
$fn(\forall x.\mathcal{A})$	$\triangleq$	$fn(\mathcal{A})$	$fv(\forall x.\mathcal{A})$	$\triangleq$	$fv(\mathcal{A})\text{-}\{x\}$
$fn(\mathcal{A}_1,...,\mathcal{A}_k)$	$\triangleq$	$fn(\mathcal{A}_1) \cup ... \cup fn(\mathcal{A}_k)$	$fv(\mathcal{A}_1,...,\mathcal{A}_k)$	$\triangleq$	$fv(\mathcal{A}_1) \cup ... \cup fv(\mathcal{A}_k)$

A formula $\mathcal{A}$ is closed if $fv(\mathcal{A}) = \emptyset$. Substitution $\mathcal{A}\{\eta{\leftarrow}\mu\}$ of a name or variable μ for another name or variable η in a formula $\mathcal{A}$, is defined in the usual way. We identify formulas up to renaming of bound variables, that is, we assume the identity $\forall x.\mathcal{A} = \forall y.\mathcal{A}\{x\leftarrow y\}$, where $y \notin fv(\mathcal{A})$. We often write $\eta[]$ for $\eta[\mathbf{0}]$, $\mathcal{A}^{\mathbf{F}}$ for $\mathcal{A}\triangleright\mathbf{F}$, and $\mathcal{A}^{\neg}$ for $\neg\mathcal{A}$.

2.3 Satisfaction

The satisfaction relation $P \vDash \mathcal{A}$ means that the process P satisfies the closed formula $\mathcal{A}$. The definition of satisfaction is based heavily on the structural congruence relation. The satisfaction relation is defined inductively in the following tables, where Π is the sort of processes, Φ is the sort of formulas, ϑ is the sort of variables, and Λ is the sort of names. We use similar syntax for logical connectives at the meta-level and object-level, but this is unambiguous.

The meaning of the temporal modality is given by reductions in the operational semantics of the ambient calculus. For the spatial modality, we need the following definitions. The relation $P\downarrow P'$ indicates that P contains P' within exactly one level of nesting. Then, $P\downarrow^* P'$ is the reflexive and transitive closure of the previous relation, indicating that P contains P' at some nesting level. Note that P' constitutes the entire contents of an enclosed ambient.

$P\downarrow P'$ iff $\exists n, P''.\ P \equiv n[P'] \mid P''$

$\downarrow^*$ is the reflexive and transitive closure of $\downarrow$

Satisfaction

$\forall P\in\Pi.$	$P \vDash \mathbf{T}$		
$\forall P\in\Pi, \mathcal{A}\in\Phi.$	$P \vDash \neg\mathcal{A}$	$\triangleq$	$\neg\, P \vDash \mathcal{A}$
$\forall P\in\Pi, \mathcal{A},\mathcal{B}\in\Phi.$	$P \vDash \mathcal{A}\vee\mathcal{B}$	$\triangleq$	$P \vDash \mathcal{A} \vee P \vDash \mathcal{B}$
$\forall P\in\Pi.$	$P \vDash \mathbf{0}$	$\triangleq$	$P \equiv \mathbf{0}$
$\forall P\in\Pi, \mathcal{A},\mathcal{B}\in\Phi.$	$P \vDash \mathcal{A} \mid \mathcal{B}$	$\triangleq$	$\exists P',P''\in\Pi.\ P \equiv P'\mid P'' \wedge P' \vDash \mathcal{A} \wedge P'' \vDash \mathcal{B}$
$\forall P\in\Pi, \mathcal{A},\mathcal{B}\in\Phi.$	$P \vDash \mathcal{A}\triangleright\mathcal{B}$	$\triangleq$	$\forall P'\in\Pi.\ P' \vDash \mathcal{A} \Rightarrow P\mid P' \vDash \mathcal{B}$
$\forall P\in\Pi, n\in\Lambda, \mathcal{A}\in\Phi.$	$P \vDash n[\mathcal{A}]$	$\triangleq$	$\exists P'\in\Pi.\ P \equiv n[P'] \wedge P' \vDash \mathcal{A}$
$\forall P\in\Pi, \mathcal{A}\in\Phi.$	$P \vDash \mathcal{A}@n$	$\triangleq$	$n[P] \vDash \mathcal{A}$
$\forall P\in\Pi, n\in\Lambda, \mathcal{A}\in\Phi.$	$P \vDash n®\mathcal{A}$	$\triangleq$	$\exists P'\in\Pi.\ P \equiv (\nu n)P' \wedge P' \vDash \mathcal{A}$
$\forall P\in\Pi, \mathcal{A}\in\Phi.$	$P \vDash \mathcal{A}\oslash n$	$\triangleq$	$(\nu n)P \vDash \mathcal{A}$
$\forall P\in\Pi, \mathcal{A}\in\Phi.$	$P \vDash \Diamond\mathcal{A}$	$\triangleq$	$\exists P'\in\Pi.\ P\rightarrow^* P' \wedge P' \vDash \mathcal{A}$
$\forall P\in\Pi, \mathcal{A}\in\Phi.$	$P \vDash ✧\mathcal{A}$	$\triangleq$	$\exists P'\in\Pi.\ P\downarrow^* P' \wedge P' \vDash \mathcal{A}$
$\forall P\in\Pi, x\in\vartheta, \mathcal{A}\in\Phi.$	$P \vDash \forall x.\mathcal{A}$	$\triangleq$	$\forall m\in\Lambda.\ P \vDash \mathcal{A}\{x\leftarrow m\}$

Again, all these logical connectives are described and discussed in [7], except for revelation and its adjunct, which are the subject of Section 3.

Remark: Given our policy of identifying formulas up to the renaming of bound variables, we need to check that satisfaction is well defined with respect to the equation $\forall x.\mathcal{A} = \forall y.\mathcal{A}\{x\leftarrow y\}$, where $y \notin fv(\mathcal{A})$. We need to show for all processes P, formulas $\mathcal{A}$, and variables x and y such that $y \notin fv(\mathcal{A})$ that $P \vDash \forall x.\mathcal{A}$ if and only if $P \vDash \forall y.\mathcal{A}\{x\leftarrow y\}$. By definition, $P \models \forall y.\mathcal{A}\{x\leftarrow y\}$ if and only if $\forall m\in\Lambda.\ P \models \mathcal{A}\{x\leftarrow y\}\{y\leftarrow m\}$. Since $y \notin fv(\mathcal{A})$, we have $\mathcal{A}\{x\leftarrow y\}\{y\leftarrow m\} = \mathcal{A}\{x\leftarrow m\}$. Therefore, $P \models \forall y.\mathcal{A}\{x\leftarrow y\}$ if and only if $\forall m\in\Lambda.\ P \models \mathcal{A}\{x\leftarrow m\}$ This is the definition of satisfaction for $\forall x.\mathcal{A}$. So it follows that $y \notin fv(\mathcal{A})$ implies that $P \models \forall x.\mathcal{A}$ if and only if $P \models \forall y.\mathcal{A}\{x\leftarrow y\}$. □

Fundamental Lemmas

The following lemmas are crucial in what follows.

2-1 Lemma (Satisfaction is up to ≡)

$(P \models \mathcal{A} \wedge P \equiv P') \Rightarrow P' \models \mathcal{A}$

Proof

A simple induction on the structure of $\mathcal{A}$.

□

2-2 Lemmas (Inversion)

(1) $P \equiv Q \ \Rightarrow\ fn(P) = fn(Q)$

(2) $(\nu n)P \equiv \mathbf{0} \ \Rightarrow\ P \equiv \mathbf{0}$

(3) $(\nu n)P \equiv m[Q] \ \Rightarrow\ \exists R\in\Pi.\ P \equiv m[R] \wedge Q \equiv (\nu n)R$ (for $n \neq m$)

(4) $(\nu n)P \equiv Q' \mid Q'' \ \Rightarrow\ \exists R',R''\in\Pi.\ P \equiv R' \mid R'' \wedge Q' \equiv (\nu n)R' \wedge Q'' \equiv (\nu n)R''$

□

See [8] for proofs of these lemmas.

Remark. It is not true that $(\nu n)P \equiv (\nu n)Q$ implies $P \equiv Q$. Take $P = n[]$ and $Q = (\nu n)n[]$; then $(\nu n)n[] \equiv (\nu n)(\nu n)n[]$ but $n[] \not\equiv (\nu n)n[]$. □

2-3 Lemma (Fresh renaming preserves ≡)

Consider any process P and names m, m', with $m' \notin fn(P)$. For all P', if $P \equiv P'$ then $m' \notin fn(P')$ and $P\{m\leftarrow m'\} \equiv P'\{m\leftarrow m'\}$. Moreover, for all Q, if $P\{m\leftarrow m'\} \equiv Q$ then there is a P' with $P \equiv P'$, $m' \notin fn(P)$ and $Q = P'\{m\leftarrow m'\}$.

□

2-4 Lemma (Fresh renaming preserves ⊨)

For all closed formulas $\mathcal{A}$, processes P, and names m, m',
if $m' \notin fn(P) \cup fn(\mathcal{A})$ then $P \models \mathcal{A} \Leftrightarrow P\{m\leftarrow m'\} \models \mathcal{A}\{m\leftarrow m'\}$.

Proof

The proof is an extension of the proof of the analogous property for our earlier modal logic [7]. If $m=m'$ the lemma holds trivially, so we may assume that $m\neq m'$. The proof is by induction on the number of symbols in the closed formula $\mathcal{A}$. The number of symbols in a formula is unchanged by substituting a name for a variable or another name. Consider an arbitrary process P, and any names m and m'. We show only the cases for revelation and hiding. The cases for the other constructs are the same as in our earlier proof.

In the case for revelation, $\mathcal{A} = n \circledR \mathcal{B}$, we prove each direction of the following separately, assuming that $m' \notin fn(P) \cup fn(n \circledR \mathcal{B})$ (and hence $m' \neq n$).

$P \models n \circledR \mathcal{B} \Leftrightarrow P\{m \leftarrow m'\} \models (n \circledR \mathcal{B})\{m \leftarrow m'\}$.

($\Rightarrow$) Assume $P \models n \circledR \mathcal{B}$, that is, there is P' such that $P \equiv (\nu n)P'$ and $P' \models \mathcal{B}$. By Lemma 2-2(1), $P \equiv (\nu n)P'$ implies $n \notin fn(P)$ and $m' \notin fn(P')$. By Lemma 2-3, $m' \notin fn(P)$ and $P \equiv (\nu n)P'$ implies $P\{m \leftarrow m'\} \equiv ((\nu n)P')\{m \leftarrow m'\}$. Since $m' \notin fn(P') \cup fn(\mathcal{B})$, the induction hypothesis implies that $P'\{m \leftarrow m'\} \models \mathcal{B}\{m \leftarrow m'\}$.

Next, suppose that $m=n$. We get that $(\nu m')(P'\{m \leftarrow m'\}) \models m' \circledR (\mathcal{B}\{m \leftarrow m'\})$. Since $m' \notin fn(P')$ and $m=n$ we have $(\nu m')(P'\{m \leftarrow m'\}) = (\nu m)P' = ((\nu n)P')\{m \leftarrow m'\}$. Moreover, $m' \circledR (\mathcal{B}\{m \leftarrow m'\}) = (n \circledR \mathcal{B})\{m \leftarrow m'\}$. Hence, $((\nu n)P')\{m \leftarrow m'\} \models (n \circledR \mathcal{B})\{m \leftarrow m'\}$. By Lemma 2-1, $P\{m \leftarrow m'\} \equiv ((\nu n)P')\{m \leftarrow m'\}$ implies $P\{m \leftarrow m'\} \models (n \circledR \mathcal{B})\{m \leftarrow m'\}$.

Otherwise, suppose that $m \neq n$. We get that $(\nu n)(P'\{m \leftarrow m'\}) \models n \circledR (\mathcal{B}\{m \leftarrow m'\})$. Since $m \neq n$ and $m' \neq n$ we have $(\nu n)(P'\{m \leftarrow m'\}) = ((\nu n)P')\{m \leftarrow m'\}$. Since $m \neq n$ we have $n \circledR (\mathcal{B}\{m \leftarrow m'\}) = (n \circledR \mathcal{B})\{m \leftarrow m'\}$. Hence, $((\nu n)P')\{m \leftarrow m'\} \models (n \circledR \mathcal{B})\{m \leftarrow m'\}$. By Lemma 2-1, $P\{m \leftarrow m'\} \equiv ((\nu n)P')\{m \leftarrow m'\}$ implies $P\{m \leftarrow m'\} \models (n \circledR \mathcal{B})\{m \leftarrow m'\}$.

($\Leftarrow$) Assume $P\{m \leftarrow m'\} \models (n \circledR \mathcal{B})\{m \leftarrow m'\}$.

First, suppose that $m=n$. By assumption, there is P' such that $P\{m \leftarrow m'\} \equiv (\nu m')P'$ and $P' \models \mathcal{B}\{m \leftarrow m'\}$. By Lemma 2-2(1), $P\{m \leftarrow m'\} \equiv (\nu m')P'$ implies that $m' \notin fn(P\{m \leftarrow m'\})$ and $m \notin fn((\nu m')P')$. Since $m \neq m'$ we get that $m \notin fn(P) \cup fn(P')$ and $P \equiv (\nu m')P'$. By induction hypothesis, $m \notin fn(P') \cup fn(\mathcal{B}\{m \leftarrow m'\})$ implies that $P'\{m' \leftarrow m\} \models \mathcal{B}\{m \leftarrow m'\}\{m' \leftarrow m\}$, that is, $P'\{m' \leftarrow m\} \models \mathcal{B}$. By definition of satisfaction, $(\nu m)(P'\{m' \leftarrow m\}) \models m \circledR \mathcal{B}$, that is, $(\nu m')P' \models m \circledR \mathcal{B}$. By Lemma 2-1, $P \equiv (\nu m')P'$ implies $P \models m \circledR \mathcal{B}$, that is, $P \models n \circledR \mathcal{B}$.

Second, suppose that $m \neq n$. By assumption, there is P' such that $P\{m \leftarrow m'\} \equiv (\nu n)P'$ and $P' \models \mathcal{B}\{m \leftarrow m'\}$. By Lemma 2-2(1), $P\{m \leftarrow m'\} \equiv (\nu n)P'$ implies that $m \notin fn((\nu n)P')$. Since $m \neq n$, $m \notin fn(P')$. By induction hypothesis, $m \notin fn(P') \cup fn(\mathcal{B}\{m \leftarrow m'\})$ implies that $P'\{m' \leftarrow m\} \models \mathcal{B}\{m \leftarrow m'\}\{m' \leftarrow m\}$, that is, $P'\{m' \leftarrow m\} \models \mathcal{B}$. By definition of satisfaction, $(\nu n)(P'\{m' \leftarrow m\}) \models n \circledR \mathcal{B}$. By Lemma 2-3, $m' \notin fn(P\{m \leftarrow m'\})$ and $P\{m \leftarrow m'\}\{m' \leftarrow m\} \equiv ((\nu n)P')\{m' \leftarrow m\}$ implies $P\{m \leftarrow m'\}\{m' \leftarrow m\} \equiv ((\nu n)P')\{m' \leftarrow m\}$. Hence from $m' \notin fn(P)$, $m \neq n$, and $n \neq m'$ we can calculate $P = P\{m \leftarrow m'\}\{m' \leftarrow m\} \equiv ((\nu n)P')\{m' \leftarrow m\} = (\nu n)(P'\{m' \leftarrow m\})$. By Lemma 2-1, this and $(\nu n)(P'\{m' \leftarrow m\}) \models n \circledR \mathcal{B}$ imply $P \models n \circledR \mathcal{B}$.

In the case for hiding, $\mathcal{A} = \mathcal{B} \oslash n$, we prove the following directly, where $m' \notin fn(P) \cup fn(\mathcal{B} \oslash n)$ (and hence $m' \neq n$).

$P \models \mathcal{B} \oslash n \Leftrightarrow P\{m \leftarrow m'\} \models (\mathcal{B} \oslash n)\{m \leftarrow m'\}$.

First, suppose that $m=n$. By definition, $P \models \mathcal{B} \oslash n \Leftrightarrow (\nu n)P \models \mathcal{B}$. By induction hypothesis, $m' \notin fn((\nu n)P) \cup fn(\mathcal{B})$ implies $(\nu n)P \models \mathcal{B} \Leftrightarrow ((\nu n)P)\{n \leftarrow m'\} \models \mathcal{B}\{n \leftarrow m'\}$. We have $((\nu n)P)\{n \leftarrow m'\} = (\nu n)P = (\nu m')(P\{m \leftarrow m'\})$, since $m' \notin fn(P)$ and $m=n$. By definition, $(\nu m')(P\{m{\leftarrow}m'\}) \vDash \mathcal{B}\{n{\leftarrow}m'\} \Leftrightarrow P\{m{\leftarrow}m'\} \vDash \mathcal{B}\{n{\leftarrow}m'\}\oslash m'$. From $m{=}n$ we have $\mathcal{B}\{n{\leftarrow}m'\}\oslash m' = (\mathcal{B}\oslash n)\{m{\leftarrow}m'\}$.

Second, suppose that $m{\neq}n$. By definition, $P \vDash \mathcal{B}\oslash n \Leftrightarrow (\nu n)P \vDash \mathcal{B}$. By induction hypothesis, $m' \notin fn((\nu n)P){\cup}fn(\mathcal{B})$ implies $(\nu n)P \vDash \mathcal{B} \Leftrightarrow ((\nu n)P)\{m{\leftarrow}m'\} \vDash \mathcal{B}\{m{\leftarrow}m'\}$. We have $((\nu n)P)\{m{\leftarrow}m'\} = (\nu n)(P\{m{\leftarrow}m'\})$, since $m{\neq}n$ and $m'{\neq}n$. By definition, $(\nu n)(P\{m{\leftarrow}m'\}) \vDash \mathcal{B}\{m{\leftarrow}m'\} \Leftrightarrow P\{m{\leftarrow}m'\} \vDash \mathcal{B}\{m{\leftarrow}m'\}\oslash n$. From $m{\neq}n$ we have $\mathcal{B}\{m{\leftarrow}m'\}\oslash n = (\mathcal{B}\oslash n)\{m{\leftarrow}m'\}$.

$\square$

2.4 Validity

Valid Formulas, Sequents, and Rules

A closed formula is valid when it is satisfied by all processes. A general formula is valid when it is valid under any closed instantiation of its free variables with names.

More precisely, if $fv(\mathcal{A})=\{x_1, ..., x_k\}$ are the free variables of $\mathcal{A}$ and $\varphi\in\vartheta\to\Lambda$ is a substitution of names for variables such that $dom(\varphi)\supseteq fv(\mathcal{A})$, then we write $\mathcal{A}_\varphi$ for $\mathcal{A}\{x_1{\leftarrow}\varphi(x_1), ..., x_k{\leftarrow}\varphi(x_k)\}$, and we define:

Valid Formulas

$$\boldsymbol{vld}(\mathcal{A})_\varphi \triangleq \forall P{\in}\Pi.\ P \vDash \mathcal{A}_\varphi \qquad \text{for } \varphi\in\vartheta\to\Lambda \text{ with } dom(\varphi)\supseteq fv(\mathcal{A})$$

$$\boldsymbol{vld}(\mathcal{A}) \triangleq \forall\varphi{\in}fv(\mathcal{A})\to\Lambda.\ \boldsymbol{vld}(\mathcal{A})_\varphi$$

We use validity for interpreting logical inference rules, as described in the following tables. We use a linearized notation for inference rules, where the usual horizontal bar separating antecedents from consequents is written '$\}$' in-line, and ';' is used to separate antecedents.

Sequents are interpreted as follows. A simple sequent $\mathcal{A} \vdash \mathcal{B}$ is interpreted as the validity of the formula $\mathcal{A}{\Rightarrow}\mathcal{B}$. Sequents with conditions about disjointness of variables, or disjointness of variables from names, are reduced to simple sequents, as described below. Note that, as discussed in [7], equality of names $\eta{=}\mu$ is definable in the logic as $\eta[\mathbf{T}]@\mu$.

Sequents

$$\mathcal{A} \vdash \mathcal{B} \triangleq \boldsymbol{vld}(\mathcal{A}{\Rightarrow}\mathcal{B})$$

$$\mathcal{A} \vdash \mathcal{B}\ (\eta_1{\neq}\mu_1, ..., \eta_n{\neq}\mu_n) \triangleq (\eta_1{\neq}\mu_1 \wedge ... \wedge \eta_n{\neq}\mu_n \wedge \mathcal{A}) \vdash \mathcal{B}$$

$$\mathcal{A} \dashv\vdash \mathcal{B}\ (\Xi) \triangleq (\mathcal{A} \vdash \mathcal{B}\ (\Xi)) \wedge (\mathcal{B} \vdash \mathcal{A}\ (\Xi)) \quad \text{where } \Xi = \eta_1{\neq}\mu_1, ..., \eta_n{\neq}\mu_n$$

For example: $\mathcal{A} \vdash \mathcal{B}$ means $\forall\varphi{\in}fv(\mathcal{A}{\Rightarrow}\mathcal{B})\to\Lambda.\ \forall P{\in}\Pi.\ P \vDash \mathcal{A}_\varphi \Rightarrow P \vDash \mathcal{B}_\varphi$. To be precise, a given sequent cannot contain instances of the metavariables η and μ, but it may contain either a name or a variable where indicated by η and μ.

Logical rules are interpreted as follows, where $\mathcal{S}$ are sequents (any of the three forms above, including sequents with side conditions and double sequents):

Rules

$\mathcal{S}_1; ...; \mathcal{S}_n \;\rangle\; \mathcal{S}_0 \;\triangleq\; (\mathcal{S}_1 \wedge ... \wedge \mathcal{S}_n) \Rightarrow \mathcal{S}_0$

$\mathcal{S}_1 \;\langle\rangle\; \mathcal{S}_2 \;\triangleq\; \mathcal{S}_1 \;\rangle\; \mathcal{S}_2 \;\wedge\; \mathcal{S}_2 \;\rangle\; \mathcal{S}_1$

The definition of validity for formulas with free variables allows us to handle quantification over names. We obtain the validity of the following standard rules for the universal quantifier, and for the definable existential quantifier:

Quantification

(∀ L)	$\mathcal{A}\{x{\leftarrow}\eta\} \vdash \mathcal{B} \;\rangle\; \forall x.\mathcal{A} \vdash \mathcal{B}$	where η is a name or a variable
(∀ R)	$\mathcal{A} \vdash \mathcal{B} \;\rangle\; \mathcal{A} \vdash \forall x.\mathcal{B}$	where $x \notin fv(\mathcal{A})$
(∃ L)	$\mathcal{A} \vdash \mathcal{B} \;\rangle\; \exists x.\mathcal{A} \vdash \mathcal{B}$	where $x \notin fv(\mathcal{B})$
(∃ R)	$\mathcal{A} \vdash \mathcal{B}\{x{\leftarrow}\eta\} \;\rangle\; \mathcal{A} \vdash \exists x.\mathcal{B}$	where η is a name or a variable

***Remark:* (∀ R).** The distinction between variables and names in formulas, and the use of variables (as opposed to names) in quantification is crucial for (∀ R). The version of (∀ R) with names instead of variables:

$\mathcal{A} \vdash \mathcal{B} \;\rangle\; \mathcal{A} \vdash \forall n.\mathcal{B}$ where $n \notin fn(\mathcal{A})$,

is not sound. Consider the valid sequent $m[\mathbf{T}] \vdash \neg n[\mathbf{T}]$. If quantification binders were names, then the rule (∀ R) could be used to produce $m[\mathbf{T}] \vdash \forall n.\neg n[\mathbf{T}]$, which is not valid. Since quantification binders are variables, one can only deduce $m[\mathbf{T}] \vdash \forall x.\neg n[\mathbf{T}]$. □

***Remark:* (∀ L).** The use of substitutions that admit variables, in addition to names, in (∀ L), is crucial. Otherwise, if (∀ L) is formulated as $\mathcal{A}\{x{\leftarrow}m\} \vdash \mathcal{B} \;\rangle\; \forall x.\mathcal{A} \vdash \mathcal{B}$, there does not seem to be any way to derive, for example:

$\mathcal{A} \vdash \mathcal{B} \;\rangle\; \forall x.\mathcal{A} \vdash \forall x.\mathcal{B}$

which is obtained by starting from $\mathcal{A}\{x{\leftarrow}x\} \vdash \mathcal{B}$ and applying (∀ L) and then (∀ R). □

Remark: Scope of variables in rules. Free variables are independently quantified in each sequent of a rule, and not across the whole rule. (This is necessary for the soundness of (∀ R).) For example, the rule $\mathcal{A}_1 \vdash \mathcal{B}_1 \;\rangle\; \mathcal{A}_0 \vdash \mathcal{B}_0$ means: $(\forall\varphi{\in}fv(\mathcal{A}_1{\Rightarrow}\mathcal{B}_1){\rightarrow}\Lambda.\ \boldsymbol{vld}(\mathcal{A}_1{\Rightarrow}\mathcal{B}_1)_\varphi) \Rightarrow (\forall\varphi{\in}fv(\mathcal{A}_0{\Rightarrow}\mathcal{B}_0){\rightarrow}\Lambda.\ \boldsymbol{vld}(\mathcal{A}_0{\Rightarrow}\mathcal{B}_0)_\varphi)$. Therefore, there is no actual relationship between identical variables occurring on the left and on the right of $\rangle$. □

Remark: Name and Variable form for Axioms. When an axiom (a rule without antecedents) is meant to hold for both variables and names, we present it with variables, and we rely on an instantiation principle (Corollary 2-6) to derive the form with names. Conversely, an axiom can be systematically lifted from name form to variable form by a technique from [7] (section 4.2.8), which introduces side conditions about disjointness of variables. However, that technique often introduces unnecessary side conditions. □

Remark: Name and Variable form for Rules. When a full rule (with antecedents) is meant to hold for both variables and names, the situation gets subtle, particularly if related variables or names occur throughout the rule. Consider one of the rules we use later, expressed in variable and name form; both forms are valid:

(1) $x \circledR \mathcal{A} \vdash \mathcal{B} \ \} \ \mathcal{A} \vdash \mathcal{B} \oslash x$
(2) $n \circledR \mathcal{A} \vdash \mathcal{B} \ \} \ \mathcal{A} \vdash \mathcal{B} \oslash n$

In rule (1) the two x's are under separate φ quantifications, so this really means the same as $x \circledR \mathcal{A} \vdash \mathcal{B} \ \} \ \mathcal{A}\{x \leftarrow y\} \vdash \mathcal{B}\{x \leftarrow y\} \oslash y$ for a fresh y. Rule (1) by itself is not enough, because (2) is not derivable from (1) by the instantiation principle, and we need (2) for deductions that involve names. On the other hand, rule (2) by itself is not enough for deductions that involve variables. For example, from the axiom $\} \ x \circledR x \circledR \mathcal{A} \vdash x \circledR \mathcal{A}$ we may want to derive $x \circledR \mathcal{A} \vdash (x \circledR \mathcal{A}) \oslash x$ by (1), and then $\forall x.\ x \circledR \mathcal{A} \vdash \forall x.\ (x \circledR \mathcal{A}) \oslash x$. This conclusion cannot be derived if we do not start with variables in the first place, because otherwise we could never use ($\forall$ R). Therefore, we need both (1) and (2). A compact way to write these two rules is:

(3) $\eta \circledR \mathcal{A} \vdash \mathcal{B} \ \} \ \mathcal{A} \vdash \mathcal{B} \oslash \eta$

However, we should remember that this is just an abbreviation for (1) and (2). If we had two metavariables η and μ in a rule, this would represent four rules (although we have no need for this at the moment). □

Remark: Schematic Side Conditions. The disjointness side conditions, e.g.:

$\mathcal{A} \vdash \mathcal{B}\ (x{\neq}y) \ \ \triangleq \ \ (x{\neq}y \wedge \mathcal{A}) \vdash \mathcal{B}$

are interpreted within the logic, and always impose restrictions on the *free* variables or names of a sequent. They really express restrictions on the allowable instantiations of free variables to names, and only indirectly on disjointness of variables. A different kind of side condition is *schematic*; it can impose restrictions on bound variables, and it arises uniquely from the side condition to the rule ($\forall$ R): $x \notin fv(\mathcal{A})$, which talks about an actual variable, and not about its allowable instantiations. These schematic side conditions restrict, metatheoretically, the legal instances of a rule, and we always prefixed them by "where". In some cases a schematic side condition can look very similar to a disjointness side condition, and this can be a bit confusing. For example, consider the following derivable sequent (an instance of a derivable rule we later call ($\circledR$ $\forall$)), where y is free and x is bound:

$\} \ y \circledR \forall x.x[] \vdash \forall x.y \circledR x[] \quad$ where $x \neq y$

To explain what this side condition means, we examine the derivation, beginning with:

$\}_{(\mathrm{Id})} \ x[] \vdash x[] \ = \ x[]\{x \leftarrow x\} \vdash x[]$
$\}_{(\forall\ \mathrm{L})} \ \forall x.x[] \vdash x[]$

At this point we could legally apply ($\circledR$ $\vdash$) using x, in the following way:

$\}_{(\circledR\ \vdash)} \ x \circledR \forall x.x[] \vdash x \circledR x[]$

However, we should next apply ($\forall$ R) to put $\forall x$ on the right hand side, but this is now blocked by its side condition, because x is free on the left hand side. So, instead, we are forced to first apply ($\circledR$ $\vdash$) with a different variable y. The ($\forall$ R) side condition $x \notin fv(y \circledR \forall x.x[])$ then reduces to the side condition $x \neq y$ of the rule:

$\}_{(\circledR\ \vdash)} \ y \circledR \forall x.x[] \vdash y \circledR x[] \quad$ where $x \neq y$
$\}_{(\forall\ \mathrm{R})} \ y \circledR \forall x.\mathcal{A} \vdash \forall x.y \circledR \mathcal{A} \quad$ where $x \neq y$

Note that, for two free variables x,y, a disjointness side condition is stronger than a schematic side condition: we could have "where $x \neq y$" satisfied by taking x to be literally a different variable from y, but they could both be instantiated to the same name n, thereby violating a "$(x \neq y)$" side condition. □

Instantiation Principle

An instantiation principle follows from the definition of validity.

2-5 Proposition (Instantiation)

$\boldsymbol{vld}(\mathcal{A}) \Rightarrow \boldsymbol{vld}(\mathcal{A}\{x\leftarrow n\})$

Proof

Assume $\boldsymbol{vld}(\mathcal{A})$; that is, assume $\forall \varphi \in fv(\mathcal{A}) \to \Lambda.\ \forall P \in \Pi.\ P \vDash \mathcal{A}_\varphi$. Take any $\psi \in fv((\mathcal{A})\{x\leftarrow n\}) \to \Lambda$ and any $P \in \Pi$. If $x \in fv(\mathcal{A})$, by instantiating the assumption with $\varphi = \psi\{x\leftarrow n\}$ we have $P \vDash \mathcal{A}_{\psi\{x\leftarrow n\}}$, which is the same as $\mathcal{A}\{x\leftarrow n\}_\psi$, since $x \notin dom(\psi)$. If $x \notin fv(\mathcal{A})$, by instantiating the first assumption with $\varphi = \psi$ we have $P \vDash \mathcal{A}_\psi$, which is the same as $\mathcal{A}\{x\leftarrow n\}_\psi$. In both cases, we have shown that $\forall \psi \in fv((\mathcal{A})\{x\leftarrow n\}) \to \Lambda.\ \forall P \in \Pi.\ P \vDash \mathcal{A}\{x\leftarrow n\}_\psi$, that is, $\boldsymbol{vld}(\mathcal{A}\{x\leftarrow n\})$.

□

2-6 Corollary (Instantiation Principle)

Let $\mathcal{S}$ be a one-directional sequent, then:

(Inst) $\mathcal{S} \between \mathcal{S}\{x\leftarrow n\}$

Proof

Let $\mathcal{S} = \mathcal{A} \vdash \mathcal{B}$. Assume $\mathcal{S}$ is valid, that is $\boldsymbol{vld}(\mathcal{A} \Rightarrow \mathcal{B})$. By Proposition 2-5, we have $\boldsymbol{vld}((\mathcal{A} \Rightarrow \mathcal{B})\{x\leftarrow n\})$, that is $\mathcal{S}\{x\leftarrow n\}$.

□

Substitution Principle

Let $\mathcal{B}\{-\}$ be a formula with a set of formula holes, indicated by $-$, and let $\mathcal{B}\{\mathcal{A}\}$ denote the formula obtained by filling those holes with the formula $\mathcal{A}$, after renaming the bound variables of $\mathcal{B}$ so they do not capture free variables of $\mathcal{A}$. For any mapping $\varphi \in \vartheta \to \Lambda$, we have $\mathcal{B}\{-\}_\varphi = \mathcal{B}_\varphi\{-\}$ and $\mathcal{B}\{\mathcal{A}\}_\varphi = \mathcal{B}_\varphi\{\mathcal{A}_\varphi\}$.

2-7 Lemma (Substitution)

$\boldsymbol{vld}(\mathcal{A}' \Leftrightarrow \mathcal{A}'') \Rightarrow \boldsymbol{vld}(\mathcal{B}\{\mathcal{A}'\} \Leftrightarrow \mathcal{B}\{\mathcal{A}''\})$

Proof Outline

By induction on the structure of $\mathcal{B}\{-\}$, with induction hypothesis:

$\forall \mathcal{A}',\mathcal{A}'' \in \Phi.\ \boldsymbol{vld}(\mathcal{A}' \Leftrightarrow \mathcal{A}'') \Rightarrow \boldsymbol{vld}(\mathcal{B}\{\mathcal{A}'\} \Leftrightarrow \mathcal{B}\{\mathcal{A}''\})$, that is:

$\forall \mathcal{A}',\mathcal{A}'' \in \Phi.\ (\forall \varphi \in fv(\mathcal{A}',\mathcal{A}'') \to \Lambda.\ \forall P \in \Pi.\ P \vDash \mathcal{A}'_\varphi \Leftrightarrow P \vDash \mathcal{A}''_\varphi)$

$\Rightarrow (\forall \varphi \in fv(\mathcal{B}\{\mathcal{A}'\},\mathcal{B}\{\mathcal{A}''\}) \to \Lambda.\ \forall P \in \Pi.\ P \vDash \mathcal{B}\{\mathcal{A}'\}_\varphi \Leftrightarrow P \vDash \mathcal{B}\{\mathcal{A}''\}_\varphi)$.

□

2-8 Corollary (Substitution Principle)

(Subst) $\mathcal{A}' \dashv\vdash \mathcal{A}'' \;\wr\; \mathcal{B}\{\mathcal{A}'\} \dashv\vdash \mathcal{B}\{\mathcal{A}''\}$

Proof

Assume $\mathcal{A}' \dashv\vdash \mathcal{A}''$, that is $\boldsymbol{vld}(\mathcal{A}' \Leftrightarrow \mathcal{A}'')$. By Lemma 2-7, $\boldsymbol{vld}(\mathcal{B}\{\mathcal{A}'\} \Leftrightarrow \mathcal{B}\{\mathcal{A}''\})$, that is $\mathcal{B}\{\mathcal{A}'\} \dashv\vdash \mathcal{B}\{\mathcal{A}''\}$.

□

Case Analysis Principle

A case analysis principle is useful for proofs involving equality and inequality; inequalities often occur as side-conditions of primitive and derived rules.

2-9 Definition (Classical Predicates)

A predicate $\mathcal{A}$ is classical iff $\forall\varphi\in fv(\mathcal{A})\to\Lambda.\ \{P \,\|\, P \vDash \mathcal{A}_\varphi\} \in \{\Pi, \emptyset\}$.

□

Remark. **T**, **F**, and $\eta=\mu$ are classical predicates. So is $\mathcal{A}^{\mathbf{F}}$, for any $\mathcal{A}$ (meaning that $\mathcal{A}$ is unsatisfiable). So is the conjunction, disjunction, and negation of classical predicates. □

2-10 Proposition (Case Analysis for Classical Predicates)

Let $\mathcal{A}$ be a classical predicate. Then:

$\boldsymbol{vld}(\mathcal{B}\{\mathbf{T}\}) \wedge \boldsymbol{vld}(\mathcal{B}\{\mathbf{F}\}) \Rightarrow \boldsymbol{vld}(\mathcal{B}\{\mathcal{A}\})$

Proof

Assume $\boldsymbol{vld}(\mathcal{B}\{\mathbf{T}\}) \wedge \boldsymbol{vld}(\mathcal{B}\{\mathbf{F}\})$. Take any $\varphi\in fv(\mathcal{B}\{\mathcal{A}\})\to\Lambda$ and $P\in\Pi$. By assumption we have $P \vDash \mathcal{B}_\varphi\{\mathbf{T}\}$ and $P \vDash \mathcal{B}_\varphi\{\mathbf{F}\}$. Since $\mathcal{A}$ is classical, we have also that $\{Q \,\|\, Q \vDash \mathcal{A}_\varphi\} \in \{\Pi, \emptyset\}$. Consider the case where $\{Q \,\|\, Q \vDash \mathcal{A}_\varphi\} = \Pi$. For the closed formula $\mathcal{A}_\varphi$ we have $\boldsymbol{vld}(\mathcal{A}_\varphi \Leftrightarrow \mathbf{T})$. By Lemma 2-7, $\boldsymbol{vld}(\mathcal{B}\{\mathcal{A}_\varphi\} \Leftrightarrow \mathcal{B}\{\mathbf{T}\})$, and in particular $P \vDash \mathcal{B}_\varphi\{\mathcal{A}_\varphi\}$ iff $P \vDash \mathcal{B}_\varphi\{\mathbf{T}\}$, hence we obtain $P \vDash \mathcal{B}_\varphi\{\mathcal{A}_\varphi\}$, that is $P \vDash \mathcal{B}\{\mathcal{A}\}_\varphi$. Consider now the case where $\{Q \,\|\, Q \vDash \mathcal{A}_\varphi\} = \emptyset$. For the closed formula $\mathcal{A}_\varphi$ we have $\boldsymbol{vld}(\mathcal{A}_\varphi \Leftrightarrow \mathbf{F})$. By Lemma 2-7, $\boldsymbol{vld}(\mathcal{B}\{\mathcal{A}_\varphi\} \Leftrightarrow \mathcal{B}\{\mathbf{F}\})$, and in particular $P \vDash \mathcal{B}_\varphi\{\mathcal{A}_\varphi\}$ iff $P \vDash \mathcal{B}_\varphi\{\mathbf{F}\}$, hence we obtain $P \vDash \mathcal{B}_\varphi\{\mathcal{A}_\varphi\}$, that is $P \vDash \mathcal{B}\{\mathcal{A}\}_\varphi$. In both cases, we have shown that $\forall\varphi\in fv(\mathcal{B}\{\mathcal{A}\})\to\Lambda.\ \forall P\in\Pi.\ P \vDash \mathcal{B}\{\mathcal{A}\}_\varphi$, that is $\boldsymbol{vld}(\mathcal{B}\{\mathcal{A}\})$.

□

2-11 Corollary (Case Analysis Principle)

Let $\mathcal{S}\{-\}$ be a one-directional sequent with a set of formula holes, and $\mathcal{A}$ be a classical predicate. Then:

(Case Analysis) $\mathcal{S}\{\mathbf{T}\}; \mathcal{S}\{\mathbf{F}\} \;\wr\; \mathcal{S}\{\mathcal{A}\}$

Proof

Let $\mathcal{S}\{-\}$ be $\mathcal{B}'\{-\} \vdash \mathcal{B}''\{-\}$. Assume $\mathcal{S}\{\mathbf{T}\}$ and $\mathcal{S}\{\mathbf{F}\}$, that is $\boldsymbol{vld}((\mathcal{B}' \Rightarrow \mathcal{B}'')\{\mathbf{T}\})$ and $\boldsymbol{vld}((\mathcal{B}' \Rightarrow \mathcal{B}'')\{\mathbf{F}\})$. By Proposition 2-10, we have $\boldsymbol{vld}((\mathcal{B}' \Rightarrow \mathcal{B}'')\{\mathcal{A}\})$, that is $\mathcal{S}\{\mathcal{A}\}$.

□

3 Revelation

We now study the logical connectives $\eta®\mathcal{A}$ (*revelation*), and $\mathcal{A}⊘\eta$ (*revelation adjunct* or *hiding*). These connectives make assertions about restricted names that occur at the process level.

3.1 Satisfaction

The formula $\eta®\mathcal{A}$ is used to *reveal* a restricted name; it is read "reveal η then $\mathcal{A}$", where η is either a name (n) or the occurrence of a variable (x) that denotes a name. A process P satisfies the formula $n®\mathcal{A}$ if it is possible to pull a restricted name occurring in P to the top and rename it n, and then strip off the restriction to leave a residual process that satisfies $\mathcal{A}$.[1]

We cannot rename a top-level restricted name of P to n if n is already free in P. Therefore, a revelation formula provides a way of testing for the free names of the underlying process P, as we discuss below.

The inverse (technically, the adjunct) of revelation is called *hiding*: $\mathcal{A}⊘\eta$, which is read "hide η then $\mathcal{A}$". A process P satisfies the formula $\mathcal{A}⊘n$ if $(\nu n)P$ satisfies $\mathcal{A}$, that is, if it is possible to hide n in P and then satisfy $\mathcal{A}$. The satisfaction relation $P \vDash \mathcal{A}$ for revelation and hiding is repeated below:

Satisfaction for Revelation and Hiding

$$P \vDash n®\mathcal{A} \triangleq \exists P'{\in}\Pi.\ P \equiv (\nu n)P' \wedge P' \vDash \mathcal{A}$$
$$P \vDash \mathcal{A}⊘n \triangleq (\nu n)P \vDash \mathcal{A}$$

Here are some simple examples:

$(\nu n)n[] \vDash n®\mathbf{T}$	because $n[] \vDash \mathbf{T}$
$\mathbf{0} \vDash n®\mathbf{T}$	because $\mathbf{0} \equiv (\nu n)\mathbf{0}$ and $\mathbf{0} \vDash \mathbf{T}$
$\neg\, n[] \vDash n®\mathbf{T}$	because there is no process $(\nu n)P' \equiv n[]$
$(\nu m)m[] \vDash n®n[]$	because $(\nu m)m[] \equiv (\nu n)n[]$ and $n[] \vDash n[]$
$m[] \vDash (n®n[])⊘m$	because $(\nu m)m[] \vDash n®n[]$

Revelation gives us a way to talk about the free (or "known") names of a process. This can be embodied in a derived operator ©n, satisfied by a process P iff $n{\in}fn(P)$. Several other derived connectives can be imagined:

[1] Satisfaction is defined between a process and a *closed* formula. So, although formulas like $x®\mathcal{A}$ are allowed in general, whenever we discuss satisfaction we use formulas like $n®\mathcal{A}$.

Examples of Derived Connectives

$\copyright\eta$	$\triangleq \neg\eta\circledR\mathbf{T}$	contains η free (it is not possible to reveal η)
closed	$\triangleq \neg\exists x.\copyright x$	no free names
separate	$\triangleq \neg\exists x.\copyright x \mid \copyright x$	no shared free names
atmostfree η	$\triangleq$ *closed*$\oslash\eta$	contains at most η free

For clarity, we expand the definitions of these derived connectives:

Expanded Definitions

$\forall P{\in}\Pi.$	$P \vDash \copyright n$	iff $\neg\exists P'{\in}\Pi.\ P \equiv (\nu n)P'$	iff $n{\in}fn(P)$
$\forall P{\in}\Pi.$	$P \vDash$ *closed*	iff $\forall n{\in}\Lambda.\ \exists P'{\in}\Pi.\ P \equiv (\nu n)P'$	iff $fn(P) = \emptyset$
$\forall P{\in}\Pi.$	$P \vDash$ *separate*	iff $\neg\exists n{\in}\Lambda.\ \exists P',P''{\in}\Pi.\ P \equiv P' \mid P'' \wedge n{\in}fn(P') \wedge n{\in}fn(P'')$	
$\forall P{\in}\Pi.$	$P \vDash$ *atmostfree* n	iff $\forall m{\in}\Lambda.\ \exists P'{\in}\Pi.\ (\nu n)P \equiv (\nu m)P'$	iff $fn(P) \subseteq \{n\}$

Examples:

$n[] \vDash \copyright n$ because $\neg\exists P'{\in}\Pi.\ n[] \equiv (\nu n)P'$

$(\nu m)m[] \vDash closed$ because $\forall n{\in}\Lambda.\ (\nu m)m[] \equiv (\nu n)(\nu m)m[]$

$n[] \mid m[] \mid (\nu p)(p[] \mid p[]) \vDash separate$

3.2 Rules

Before giving our set of primitive rules of revelation and hiding, we discuss the most interesting properties of $\circledR$ and $\oslash$ that are derived in this section. In order to emphasize some symmetries, we use here a combination of primitive and derived rules,

First, the cancellation and swapping properties of double restriction, $(\nu n)(\nu n)P \equiv (\nu n)P$ and $(\nu n)(\nu m)P \equiv (\nu m)(\nu n)P$, are inherited by both $\circledR$ and $\oslash$:

$$n\circledR n\circledR\mathcal{A} \dashv\vdash n\circledR\mathcal{A}$$
$$\mathcal{A}\oslash n\oslash n \dashv\vdash \mathcal{A}\oslash n$$
$$n\circledR m\circledR\mathcal{A} \vdash m\circledR n\circledR\mathcal{A}$$
$$\mathcal{A}\oslash m\oslash n \vdash \mathcal{A}\oslash n\oslash m$$

Next, consider the combinations:

$$n\circledR(\mathcal{A}\oslash n)$$
$$(n\circledR\mathcal{A})\oslash n$$

We see easily that $P \vDash n\circledR(\mathcal{A}\oslash n)$ means that $P \vDash \mathcal{A}$ and that $n{\notin}fn(P)$, where $n{\notin}fn(P)$ can be written also as $P \vDash n\circledR\mathbf{T}$. Instead, $P \vDash (n\circledR\mathcal{A})\oslash n$ means that, although P may not satisfy $\mathcal{A}$, if we hide n in P we obtain something where we can reveal n and satisfy $\mathcal{A}$. For example, $(\nu m)n[m[]] \nvDash m\circledR m[n[]]$, but $(\nu m)n[m[]] \vDash (n\circledR m\circledR m[n[]])\oslash n$, because $(\nu n)(\nu m)n[m[]] \equiv (\nu n)(\nu m)m[n[]] \vDash n\circledR m\circledR m[n[]]$. In other words, $P \vDash (n\circledR\mathcal{A})\oslash n$ means that we can satisfy $\mathcal{A}$ by hiding the name n of P, and revealing a possibly different restricted name of P as n. We obtain the properties:

$n®(\mathcal{A}⊘n) ⊣⊢ \mathcal{A} \wedge n®\mathbf{T}$

$n®(\mathcal{A}⊘n) \vdash \mathcal{A}$	$\mathcal{A} \vdash (n®\mathcal{A})⊘n$
$n®(\mathcal{A}⊘n) \vdash \mathcal{A}⊘n$	$\mathcal{A}⊘n \vdash (n®\mathcal{A})⊘n$
$n®(\mathcal{A}⊘n) \vdash n®\mathcal{A}$	$n®\mathcal{A} \vdash (n®\mathcal{A})⊘n$

The interactions of ® and ⊘ with | are the most interesting, and the most complex. There are basically three distribution rules: distribution of ® over | in both directions (with a constraint), unrestricted distribution of ⊘ over | in one direction, and distribution of $n®((-)⊘n)$ over | in both directions.

$n®(\mathcal{A} \mid n®\mathcal{B}) ⊣⊢ n®\mathcal{A} \mid n®\mathcal{B}$
$(\mathcal{A} \mid \mathcal{B})⊘n \vdash \mathcal{A}⊘n \mid \mathcal{B}⊘n$
$n®((\mathcal{A} \mid \mathcal{B})⊘n) ⊣⊢ n®(\mathcal{A}⊘n) \mid n®(\mathcal{B}⊘n)$

The first rule embodies the scope extrusion rule, $(\nu n)(P \mid Q) \equiv ((\nu n)P) \mid Q$ if $n \notin fn(Q)$. This can be seen more clearly if we note that the side condition $n \notin fn(Q)$ is equivalent to $Q \equiv (\nu n)Q$; then the extrusion rule can be written as $(\nu n)(P \mid (\nu n)Q) \equiv ((\nu n)P) \mid ((\nu n)Q)$ with no side condition.

The second rule implies that if $(\nu n)(P \mid Q) \vDash \mathcal{A} \mid \mathcal{B}$ then it is possible to distribute the restriction so that $(\nu n)P \vDash \mathcal{A}$ and $(\nu n)Q \vDash \mathcal{B}$; this is a consequence of Lemma 2-2(4).

The last rule looks mysterious, but has a simple interpretation. According to one of the equivalences above, it can be rewritten as $(\mathcal{A} \mid \mathcal{B}) \wedge n®\mathbf{T} ⊣⊢ (\mathcal{A} \wedge n®\mathbf{T}) \mid (\mathcal{B} \wedge n®\mathbf{T})$; that is, the name n does not occur in a parallel composition iff it does not occur in either component. The right-to-left direction is actually a derivable rule.

A similar set of rules holds for distribution of ® and ⊘ over $n[-]$:

$n®m[\mathcal{A}] ⊣⊢ m[n®\mathcal{A}]$	$(n \neq m)$
$m[\mathcal{A}]⊘n ⊣⊢ m[\mathcal{A}⊘n]$	$(n \neq m)$
$n[\mathcal{A}]⊘n ⊣⊢ \mathbf{F}$	
$n®(m[\mathcal{A}]⊘n) ⊣⊢ m[n®(\mathcal{A}⊘n)]$	$(n \neq m)$

The distribution of $n®-$ over $m[-]$ (first rule) holds in both directions as long as $n \neq m$.

The distribution of $-⊘n$ over $m[-]$ (second and third rules) comes in two cases, depending on whether $n{=}m$. In each case, the right-to-left direction is derivable. From $n[\mathbf{T}]⊘n \vdash \mathbf{F}$ we can derive $n®\mathbf{T} \vdash \neg n[\mathbf{T}]$, which means that if a name n does not occur free in a process, the process cannot be a location named n.

The distribution of $n®((-)⊘n)$ over $m[-]$ (fourth rule) is derivable in both directions, from the first two rules. Again, this rule can be rewritten as $m[\mathcal{A}] \wedge n®\mathbf{T} ⊣⊢ m[\mathcal{A} \wedge n®\mathbf{T}]$ $(n \neq m)$; that is, the name n does not occur in a location iff it is distinct from the name of the location and it does not occur inside the location.

Finally, ® and ⊘ commute in one direction:

$m®(\mathcal{A}⊘n) \vdash (m®\mathcal{A})⊘n$

We now take the following set of rules as primitive, and we verify their validity in the model. The first group handles double revelation, distribution of ® over ∨, congruence of

$\circledR$ with $\vdash$, the adjunction rule connecting $\circledR$ and $\oslash$, and the rather curious but very useful fact that $\neg$ commutes with $\oslash$. The next three groups deal with the interactions of $\circledR$ and $\oslash$ with $\mathbf{0}$, $\mid$, and $n[-]$.

Axiom naming conventions. This is not a strict rule, but very often axioms are named by a sequence of connectives corresponding to a top-down path through the formula on the left-hand side of the sequent. Occasionally, different but closely related rules have the same name: this is intentional and causes little ambiguity.

3-1 Proposition (Validity: Revelation Rules)

($\circledR$)	$\} \; x\circledR x\circledR\mathcal{A} \dashv\vdash x\circledR\mathcal{A}$	
($\circledR$ $\circledR$)	$\} \; x\circledR y\circledR\mathcal{A} \vdash y\circledR x\circledR\mathcal{A}$	
($\circledR$ $\vee$)	$\} \; x\circledR(\mathcal{A} \vee \mathcal{B}) \vdash x\circledR\mathcal{A} \vee x\circledR\mathcal{A}$	
($\circledR$ $\vdash$)	$\mathcal{A} \vdash \mathcal{B} \; \} \; x\circledR\mathcal{A} \vdash x\circledR\mathcal{B}$	
($\circledR$ $\oslash$)	$\eta\circledR\mathcal{A} \vdash \mathcal{B} \; \{\} \; \mathcal{A} \vdash \mathcal{B}\oslash\eta$	
($\oslash$ $\neg$)	$\} \; (\neg\mathcal{A})\oslash x \dashv\vdash \neg(\mathcal{A}\oslash x)$	
($\oslash$ $\triangleright\mathbf{F}$)	$\} \; \mathcal{A}^{\mathbf{F}}\oslash x \dashv\vdash \mathcal{A}^{\mathbf{F}}$	
($\circledR$ $\mathbf{0}$)	$\} \; x\circledR\mathbf{0} \dashv\vdash \mathbf{0}$	
($\oslash$ $\mathbf{0}$)	$\} \; \mathbf{0}\oslash x \vdash \mathbf{0}$	
($\circledR$ $\mid$)	$\} \; x\circledR(\mathcal{A} \mid x\circledR\mathcal{B}) \dashv\vdash x\circledR\mathcal{A} \mid x\circledR\mathcal{B}$	
($\oslash$ $\mid$)	$\} \; (\mathcal{A} \mid \mathcal{B})\oslash x \vdash \mathcal{A}\oslash x \mid \mathcal{B}\oslash x$	
($\circledR$ $\oslash$ $\mid$)	$\} \; x\circledR((\mathcal{A} \mid \mathcal{B})\oslash x) \vdash x\circledR(\mathcal{A}\oslash x) \mid x\circledR(\mathcal{B}\oslash x)$	
($\circledR$ $n[]$)	$\} \; x\circledR y[\mathcal{A}] \dashv\vdash y[x\circledR\mathcal{A}]$	$(x \neq y)$
($\oslash$ $n[]$)	$\} \; y[\mathcal{A}]\oslash x \vdash y[\mathcal{A}\oslash x]$	$(x \neq y)$
($\oslash$ $n[]$)	$\} \; x[\mathcal{A}]\oslash x \vdash \mathbf{F}$	

$\square$

Remark. The converse of ($\oslash$ $\mid$) fails. Consider $\mathcal{A}\oslash n \mid \mathcal{B}\oslash n \vdash (\mathcal{A} \mid \mathcal{B})\oslash n$. We have $n[] \mid n[] \vDash (n\circledR n[])\oslash n \mid (n\circledR n[])\oslash n$, but $n[] \mid n[] \nvDash (n\circledR n[] \mid n\circledR n[])\oslash n$. $\square$

Remark. $n\circledR\mathcal{A} \wedge n\circledR\mathcal{B} \vdash n\circledR(\mathcal{A} \wedge \mathcal{B})$ fails (the converse is derivable). We have $(\nu n)(\nu n')n[n'[]] \vDash n\circledR(n'\circledR n[n'[\mathbf{0}]]) \wedge n\circledR(n'\circledR n'[n[\mathbf{0}]])$, but $(\nu n)(\nu n')n[n'[]] \nvDash n\circledR(n'\circledR n[n'[\mathbf{0}]] \wedge n'\circledR n'[n[\mathbf{0}]])$. $\square$

Remark. $n\circledR\mathcal{A} \mid n\circledR\mathcal{B} \dashv\vdash n\circledR(\mathcal{A} \mid \mathcal{B})$ fails in both directions. We have $(\nu n)(n[] \mid n[]) \vDash n\circledR(n[] \mid n[])$, but $(\nu n)(n[] \mid n[]) \nvDash n\circledR n[] \mid n\circledR n[]$. We have $(\nu n)n[] \mid (\nu n)n[] \vDash n\circledR n[] \mid n\circledR n[]$, but $(\nu n)n[] \mid (\nu n)n[] \nvDash n\circledR(n[] \mid n[])$. $\square$

Remark. $\} \; (x\circledR\mathcal{A})\oslash y \vdash x\circledR(\mathcal{A}\oslash y)$ $(x \neq y)$ fails. (The converse is derivable without side condition: see ($\oslash$ $\circledR$ $\neq$) below). For $m \neq n$, we have $(\nu m)m[] \vDash n\circledR n[]$; therefore $m[] \vDash (n\circledR n[])\oslash m$. If the rule holds, we then obtain $m[] \vDash n\circledR(n[]\oslash m)$. This means that $\exists P' \in \Pi$. $m[] \equiv (\nu n)P' \wedge P' \vDash n[]\oslash m$; that is, $(\nu n)P' \vDash n[]$, that is $\exists P'' \in \Pi$. $(\nu n)P' \equiv n[P''] \wedge P'' \equiv \mathbf{0}$, that is $(\nu n)P' \equiv n[]$. Then, from $m[] \equiv (\nu n)P'$ and $(\nu n)P' \equiv n[]$ we obtain $m[] \equiv n[]$: contradiction. $\square$

Remark. The converse of ($\oslash$ **0**), namely $\mathbf{0} \vdash \mathbf{0}\oslash x$, is derivable from ($\circledR$ **0**). ☐

Remark. Note that $n[\mathcal{A}]\oslash n \vdash \mathbf{F}$ is the same as $n[\mathbf{T}] \vdash \neg n\circledR\mathbf{T}$, i.e. if n occurs free then it cannot be revealed. ☐

From the rules that we have validated in Proposition 3-1, we can derive a large collection of facts by logical deduction, including the following:

3-2 Logical Corollaries (Case Analysis)

Let Cl be a classical predicate (typically, Cl is a side condition of the form $x \neq y$).

(**CA** $\mid \wedge$)	$\vDash (\mathit{Cl} \wedge \mathcal{A}) \mid (\mathit{Cl} \wedge \mathcal{B}) \dashv\vdash \mathit{Cl} \wedge (\mathcal{A} \mid \mathcal{B})$	
(**CA** $\mid \Rightarrow$)	$\vDash (\mathit{Cl} \Rightarrow \mathcal{A}) \mid (\mathit{Cl} \Rightarrow \mathcal{B}) \vdash \mathit{Cl} \Rightarrow (\mathcal{A} \mid \mathcal{B})$	
(**CA** $n[]\ \wedge$)	$\vDash z[\mathit{Cl} \wedge \mathcal{A}] \dashv\vdash \mathit{Cl} \wedge z[\mathcal{A}]$	where z may occur in Cl
(**CA** $\circledR \wedge$)	$\vDash z\circledR(\mathit{Cl} \wedge \mathcal{A}) \dashv\vdash \mathit{Cl} \wedge z\circledR\mathcal{A}$	where z may occur in Cl

☐

3-3 Logical Corollaries (Revelation)

($\oslash$)	$\vDash \mathcal{A}\oslash x\oslash x \dashv\vdash \mathcal{A}\oslash x$	
($\oslash\ \oslash$)	$\vDash \mathcal{A}\oslash y\oslash x \vdash \mathcal{A}\oslash x\oslash y$	
($\circledR \oslash$ R)	$\vDash \mathcal{A} \vdash (x\circledR\mathcal{A})\oslash x$	
	$\vDash x\circledR\mathcal{A} \vdash (x\circledR\mathcal{A})\oslash x$	
($\circledR \oslash$ L)	$\vDash x\circledR(\mathcal{A}\oslash x) \vdash \mathcal{A}$	
	$\vDash x\circledR(\mathcal{A}\oslash x) \vdash \mathcal{A}\oslash x$	
($\circledR \oslash \mid$)	$\vDash x\circledR((\mathcal{A} \mid \mathcal{B})\oslash x) \dashv\vdash x\circledR(\mathcal{A}\oslash x) \mid x\circledR(\mathcal{B}\oslash x)$	
($\circledR \mid \circledR$)	$\vDash x\circledR(x\circledR\mathcal{A} \mid x\circledR\mathcal{B}) \dashv\vdash x\circledR\mathcal{A} \mid x\circledR\mathcal{B}$	
	$\vDash x\circledR\mathcal{A} \mid x\circledR\mathcal{B} \dashv\vdash (x\circledR\mathcal{A} \mid x\circledR\mathcal{B})\oslash x$	
($\mid \circledR \oslash$)	$\vDash x\circledR\mathcal{A} \mid x\circledR(\mathcal{B}\oslash x) \vdash x\circledR(\mathcal{A} \mid \mathcal{B})$	
($\oslash \mid \circledR$)	$\vDash \mathcal{A}\oslash x \mid x\circledR\mathcal{B} \vdash (\mathcal{A} \mid x\circledR\mathcal{B})\oslash x$	
($\circledR \vee$)	$\vDash x\circledR(\mathcal{A} \vee \mathcal{B}) \dashv\vdash x\circledR\mathcal{A} \vee x\circledR\mathcal{B}$	
($\circledR \wedge$)	$\vDash x\circledR(\mathcal{A} \wedge \mathcal{B}) \vdash x\circledR\mathcal{A} \wedge x\circledR\mathcal{B}$	hence: $\vDash x\circledR\mathcal{A} \dashv\vdash x\circledR\mathcal{A} \wedge x\circledR\mathbf{T}$
($\circledR$ **F**)	$\vDash x\circledR\mathbf{F} \vdash \mathbf{F}$	
($\oslash$ **T**)	$\vDash \mathbf{T} \dashv\vdash \mathbf{T}\oslash x$	
($\oslash$ **F**)	$\vDash \mathbf{F}\oslash x \dashv\vdash \mathbf{F}$	
($\oslash \vee$)	$\vDash (\mathcal{A} \vee \mathcal{B})\oslash x \dashv\vdash \mathcal{A}\oslash x \vee \mathcal{B}\oslash x$	
($\oslash \wedge$)	$\vDash (\mathcal{A} \wedge \mathcal{B})\oslash x \dashv\vdash \mathcal{A}\oslash x \wedge \mathcal{B}\oslash x$	
($\oslash$ **0**)	$\vDash \mathbf{0} \vdash \mathbf{0}\oslash x$	
($\circledR \neg$**0**)	$\vDash x\circledR\neg\mathbf{0} \vdash \neg\mathbf{0}$	
($\oslash \mid$)	$\vDash \mathcal{A}\oslash x \mid x\circledR(\mathcal{B}\oslash x) \vdash (\mathcal{A} \mid \mathcal{B})\oslash x$	
	$\vDash (\mathcal{A} \mid \mathcal{B})\oslash x \vdash \mathcal{A}\oslash x \mid \mathcal{B}\oslash x$	
($\oslash \circledR$)	$\vDash \mathcal{A}\oslash x \vdash (x\circledR\mathcal{A})\oslash x$	hence: $\vDash x\circledR(\mathcal{A}\oslash x) \vdash x\circledR\mathcal{A}$
($\circledR \wedge \oslash$)	$\vDash x\circledR(\mathcal{A} \wedge \mathcal{B}\oslash x) \dashv\vdash x\circledR\mathcal{A} \wedge \mathcal{B}$	hence: $\vDash x\circledR(\mathcal{B}\oslash x) \dashv\vdash x\circledR\mathbf{T} \wedge \mathcal{B}$
($\circledR \vee \oslash$)	$\vDash x\circledR(\mathcal{A} \vee \mathcal{B}\oslash x) \vdash x\circledR\mathcal{A} \vee \mathcal{B}$	
($\oslash \vdash$)	$\mathcal{A} \vdash \mathcal{B} \vDash \mathcal{A}\oslash x \vdash \mathcal{B}\oslash x$	
($\circledR \oslash \mid$)	$\vDash x\circledR((\mathcal{A} \mid \mathcal{B})\oslash x) \dashv\vdash x\circledR(\mathcal{A}\oslash x) \mid x\circledR(\mathcal{B}\oslash x)$	

$(\circledR\ \oslash\ \mid)$	$\}\ x\circledR((\mathcal{A} \mid \mathcal{B})\oslash x) \vdash (x\circledR\mathcal{A})\oslash x \mid (x\circledR\mathcal{B})\oslash x$	
$(\circledR \wedge \mid)$	$\}\ x\circledR\mathbf{T} \wedge (\mathcal{A} \mid \mathcal{B}) \dashv\vdash (x\circledR\mathbf{T} \wedge \mathcal{A}) \mid (x\circledR\mathbf{T} \wedge \mathcal{B})$	
$(\circledR \Rightarrow \mid)$	$\}\ (x\circledR\mathbf{T}\Rightarrow\mathcal{A}) \mid (x\circledR\mathbf{T}\Rightarrow\mathcal{B}) \vdash x\circledR\mathbf{T} \Rightarrow (\mathcal{A} \mid \mathcal{B})$	
$(\oslash\ n[])$	$\}\ y[\mathcal{A}]\oslash x \dashv\vdash y[\mathcal{A}\oslash x]$	$(x \neq y)$
$(\oslash\ n[])$	$\}\ x[\mathcal{A}]\oslash x \dashv\vdash \mathbf{F}$	
$(@\ \circledR)$	$\}\ (x\circledR\mathcal{A})@x \dashv\vdash \mathbf{F}$	
$(@\ \circledR \neq)$	$\}\ (x\circledR\mathcal{A})@y \dashv\vdash x\circledR(\mathcal{A}@y)$	$(x \neq y)$
$(\circledR\ \oslash\ n[])$	$\}\ x\circledR(y[\mathcal{A}]\oslash x) \dashv\vdash y[x\circledR(\mathcal{A}\oslash x)]$	$(x \neq y)$
$(\circledR \wedge n[])$	$\}\ x\circledR\mathbf{T} \wedge y[\mathcal{A}] \dashv\vdash y[x\circledR\mathbf{T} \wedge \mathcal{A}]$	$(x \neq y)$
$(\oslash\ \circledR \neq)$	$\}\ x\circledR(\mathcal{A}\oslash y) \vdash (x\circledR\mathcal{A})\oslash y$	
$(\circledR\ \exists)$	$\}\ \exists x.y\circledR\mathcal{A} \dashv\vdash y\circledR\exists x.\mathcal{A}$	where $x \neq y$
$(\circledR\ \forall)$	$\}\ y\circledR\forall x.\mathcal{A} \vdash \forall x.y\circledR\mathcal{A}$	where $x \neq y$

□

Remark. The derived rule ($\circledR\ \neg\mathbf{0}$) says that if we reveal a restricted name and find non-**0**, then the original process is also non-**0**. That is, non-**0**-ness cannot be hidden by restriction. Consider, for example, the process $P = (\nu n)n[]$. Under many standard behavioral equivalences $\approx$ we have $P \approx \mathbf{0}$ [10]. However, we have $P \vDash n\circledR\neg\mathbf{0}$, and hence by ($\circledR\ \neg\mathbf{0}$), we have that $P \vDash \neg\mathbf{0}$. This example shows quite clearly that our logic is finer than standard behavioral equivalences, and that it can inspect the structure of restricted processes. □

3.3 Hidden-Name Quantifier: First Attempts

An Attempt with Bound Names

A *hidden-name quantifier* should be a construct of the logic that allows us to talk about restricted names in processes. The simplest definition that comes to mind is the following for the hidden-name quantifier $(\nu n)\mathcal{A}$:

$(\nu n)\mathcal{A} = (\nu m)\mathcal{A}\{n\leftarrow m\}$	if $m \notin fn(\mathcal{A})$	(α-conversion)
$P \vDash (\nu n)\mathcal{A}$	iff $\exists P' \in \Pi.\ P \equiv (\nu n)P' \wedge P' \vDash \mathcal{A}$	(satisfaction)

The α-conversion property says that n is a bound name in $(\nu n)\mathcal{A}$. The definition of satisfaction for $(\nu n)\mathcal{A}$ is identical to $n\circledR\mathcal{A}$, except for the fact that n is bound.

Unfortunately, there is a problem. Start with the valid assertion $p[] \vDash \neg n[]$. From the definition above we obtain that $(\nu n)p[] \vDash (\nu n)\neg n[]$. By α-conversion we have that $(\nu n)\neg n[] = (\nu p)\neg p[]$. So, we would expect that $(\nu n)p[] \vDash (\nu p)\neg p[]$. However, this fails, because it is not possible to find a P' such that $(\nu n)p[] \equiv (\nu p)P'$, by Lemma 2-2(1).

Therefore, α-conversion of $(\nu n)\mathcal{A}$ is not a valid equivalence in the logic, which basically contradicts the notion that n is a bound name in $(\nu n)\mathcal{A}$.

Moreover, it does not seem possible to simply accept the fact that α-conversion for $(\nu n)\mathcal{A}$ is not valid. Consider a formula such as $\forall x.(\nu n)\neg n[x[]]$. Because of the standard rules for universal quantification, it is possible to instantiate x with the name n. To avoid a name capture, we would have to α-convert $(\nu n)\neg n[x[]]$ to $(\nu n')\neg n'[x[]]$. But if α-conversion is not generally valid for $(\nu n)\mathcal{A}$, then universal instantiation is also not generally valid.

The basic problem here seems to be the notion of binding names in formulas, so we look next for hidden-name quantifiers that bind variables.

A Discriminating Property

We might now try to define a formula $(\nu x)\mathcal{B}$ to mean, informally, that "for hidden name x" (hidden in the underlying process), $\mathcal{B}$ holds. The intention is that there should be some correspondence between the binder (νx) in the formula, and a binder (νn) in a process that satisfies the formula. There are several plausible definitions. To discriminate between them, we are going to require the following property, (νx-proper), for any candidate definition of $(\nu x)\mathcal{B}$:

3-4 Property (νx-proper)

For all $n\in\Lambda$, $x\in\vartheta$, $P\in\Pi$, and closed $\mathcal{A}\in\Phi$:

$$n\notin fn(P) \wedge P \vDash (\nu x)(\mathcal{A}\{n\leftarrow x\}) \quad\Leftrightarrow\quad \exists P'\in\Pi.\ P \equiv (\nu n)P' \wedge P' \vDash \mathcal{A}.$$

Corollary: $P' \vDash \mathcal{A} \;\Rightarrow\; (\nu n)P' \vDash (\nu x)(\mathcal{A}\{n\leftarrow x\})$.

□

This property can be rewritten in logical form as $n®\mathbf{T} \wedge (\nu x)(\mathcal{A}\{n\leftarrow x\}) ⫤⊢ n®\mathcal{A}$, for all n.

Assuming (νx-proper) holds, we can already obtain, for example:

$$n[] \vDash n[] \;\Rightarrow\; (\nu n)n[] \vDash (\nu x)x[]$$
$$p[] \vDash p[] \;\Rightarrow\; (\nu n)p[] \vDash (\nu x)p[]$$

Remark. It is natural to first consider the simpler discriminating property:

$$(\nu n)P' \vDash (\nu x)(\mathcal{A}\{n\leftarrow x\}) \quad\Leftrightarrow\quad P' \vDash \mathcal{A} \qquad (\nu x\text{-}1)$$

The $\Leftarrow$ direction is equivalent to (νx-proper$\Leftarrow$). However, the $\Rightarrow$ direction is inconsistent with the fundamental Lemma 2-1. Start with $n[] \vDash n[]$. By (νx-1$\Leftarrow$) we obtain $(\nu n)n[] \vDash (\nu x)x[]$. Since $(\nu n)n[] \equiv (\nu n)(\nu n)n[]$, by Lemma 2-1 we obtain that $(\nu n)(\nu n)n[] \vDash (\nu x)x[]$. Then, by ($\nu x-1\Rightarrow$) we obtain $(\nu n)n[] \vDash n[]$, that is $(\nu n)n[] \equiv n[]$, which is contradictory by Lemma 2-2(1). The problem here is that we cannot expect a (νx) in the formula to match any (νn) in the process, but only an appropriate one. Hence the refined statement of (νx-proper). □

An Attempt with Existentials

As our first candidate for $(\nu x)\mathcal{A}$, it may seem natural to use the following definition: there exists a name x that can be revealed and such that $\mathcal{A}$ is then satisfied:

$$(\nu x)\mathcal{A} \triangleq \exists x.x®\mathcal{A}$$

If there exists an m such that the underlying process satisfies $m®\mathcal{A}$, then m is not free in the process, that is, it is fresh. This matches the idea that x should denote a fresh name.

We obtain, directly from the definitions:

$$P \vDash (\nu x)\mathcal{A} \quad\text{iff}\quad \exists m\in\Lambda.\ \exists P'\in\Pi.\ P \equiv (\nu m)P' \wedge P' \vDash \mathcal{A}\{x\leftarrow m\}$$

We can verify that property (νx-proper$\Leftarrow$) is satisfied. For any n, start with $\exists P'\in\Pi$. P

$\equiv (\nu n)P' \wedge P' \vDash \mathcal{A}$, then $n \notin fn(P)$ by Lemma 2-2(1). By definition of revelation we then have that $P \vDash n®\mathcal{A}$. Since $\mathcal{A}$ is closed, this is the same as $P \vDash (x®\mathcal{A}\{n\leftarrow x\})\{x\leftarrow n\}$. We have shown that $\exists n.\ P \vDash (x®\mathcal{A}\{n\leftarrow x\})\{x\leftarrow n\}$. By definition of existential quantification this means that $P \vDash \exists x.\ x®\mathcal{A}\{n\leftarrow x\}$, that is $P \vDash (\nu x)\mathcal{A}\{n\leftarrow x\}$.

Unfortunately, there is a serious mismatch between this definition of the hidden-name quantifier and our intuitions, because we also obtain:

$$(\nu n)n[] \vDash (\nu x)p[] \qquad \text{with } n \neq p$$
$$\text{which means: } \exists m\in\Lambda.\ \exists P'\in\Pi.\ (\nu n)n[] \equiv (\nu m)P' \wedge P' \vDash p[]$$

This assertion holds because we can choose $m = p$ and $P' = p[]$ (where p is the name that happens to be free in the formula). Instead, if m is taken to be any other name, then the assertion always fails. So, although p is "fresh" with respect to the process, we cannot equivalently take any other fresh name (with respect to the process) in order to satisfy the formula. This property seems to contradict the concept of "freshness": the assertion holds essentially because of a name clash, and should be intuitively undesirable.

More cogently, this example shows that the property (νx-proper$\Rightarrow$) fails. We have that $(\nu n)n[] \vDash (\nu x)p[]\{n\leftarrow x\}$, with $n\notin fn((\nu n)n[])$, so we should show that $\exists P'\in\Pi.\ (\nu n)n[] \equiv (\nu n)P' \wedge P' \vDash p[]$. Now, $P' \vDash p[]$ implies, by definition of $\vDash$ and of structural congruence, that $P' \equiv p[]$. Thus, by Lemma 2-2(1), $fn(P') = \{p\}$ and $fn((\nu n)P') = \{p\}$. However, $fn((\nu n)n[]) = \emptyset$. Hence, $(\nu n)n[] \equiv (\nu n)P'$ is impossible, by Lemma 2-2(1).

In conclusion, the existential definition of the hidden-name quantifier satisfies (νx-proper$\Leftarrow$), but violates (νx-proper$\Rightarrow$) because of clashes with free names in formulas.

An Attempt with Universals

Because of the problems with existentials, it may seem better to adopt a universal definition, so as to rule out accidental satisfactions due to the existence of particular names:

$$(\nu x)\mathcal{A} \triangleq \forall x.x®\mathcal{A}$$

We obtain, directly from the definitions:

$$P \vDash (\nu x)\mathcal{A} \quad \text{iff} \quad \forall m\in\Lambda.\ \exists P'\in\Pi.\ P \equiv (\nu m)P' \wedge P' \vDash \mathcal{A}\{x\leftarrow m\}$$

As a sanity check, we obtain the expected $(\nu n)n[] \vDash (\nu x)x[]$, because for any m there exists a $P' = m[]$ such that $(\nu n)n[] \equiv (\nu m)P'$ and $P' \vDash x[]\{x\leftarrow m\}$.

Moreover, this time $(\nu n)n[] \nvDash (\nu x)p[]$, because if we choose $m \neq p$ then we should show that there exists a P' such that $(\nu n)n[] \equiv (\nu m)P'$ and $P' \vDash p[]$. But $P' \vDash p[]$ implies that $P' \equiv p[]$, and then we would have $(\nu n)n[] \equiv (\nu m)p[]$, which is impossible (Lemma 2-2(1)).

In fact, we find that (νx-proper$\Rightarrow$) is satisfied in general. Assume $n\notin fn(P)$ and $P \vDash \forall x.x®(\mathcal{A}\{n\leftarrow x\})$, that is $\forall m\in\Lambda.\ \exists P'\in\Pi.\ P \equiv (\nu m)P' \wedge P' \vDash \mathcal{A}\{n\leftarrow x\}\{x\leftarrow m\}$. Then, for $m = n$, we obtain in particular that $\exists P'\in\Pi.\ P \equiv (\nu n)P' \wedge P' \vDash \mathcal{A}$.

Unfortunately, this time property (νx-proper$\Leftarrow$) fails. Consider the valid assertion $p[] \vDash \neg n[]$ with $n \neq p$; by (νx-proper$\Leftarrow$) we would obtain that $(\nu n)p[] \vDash \forall x.x®\neg x[]$. This means that $\forall m\in\Lambda.\ \exists P'\in\Pi.\ (\nu n)p[] \equiv (\nu m)P' \wedge P' \vDash \neg m[]$. In particular, for $m = p$, we obtain $\exists P'\in\Pi.\ (\nu n)p[] \equiv (\nu p)P' \wedge P' \vDash \neg p[]$. But $(\nu n)p[] \equiv (\nu p)P'$ is impossible by Lemma 2-2(1).

In conclusion, the universal definition of the hidden-name quantifier satisfies (νx-

proper⇒), but violates (νx-proper⇐) because of clashes with free names in processes.

Two Attempt with Fresh Names

A more refined attempt, then, might involve ruling out the names that are free in the formula and the processes, so that we avoid the problems above. For the simple case where $(\nu x)\mathcal{A}$ is closed, we would have:

$$(\nu x)\mathcal{A} \triangleq \exists x.\ x{\neq}n_1 \wedge ... \wedge x{\neq}n_k \wedge x\text{®}\mathbf{T} \wedge x\text{®}\mathcal{A}$$
$$\text{where } \{n_1, ..., n_k\} = fn(\mathcal{A})$$

This is in fact what we will come to eventually (suitably generalizing to open formulas). It can be read as "there exists a fresh name x (distinct from the free names of $\mathcal{A}$ and the free names of the underlying process) such that a restricted process name can be revealed as x, and then $\mathcal{A}$ can be satisfied". This definition will satisfy (νx-proper).

However, there is another plausible definition:

$$(\nu x)\mathcal{A} \triangleq \forall x.\ (x{\neq}n_1 \wedge ... \wedge x{\neq}n_k \wedge x\text{®}\mathbf{T}) \Rightarrow x\text{®}\mathcal{A}$$
$$\text{where } \{n_1, ..., n_k\} = fn(\mathcal{A})$$

which can be read "for every fresh name x (distinct from the free names of $\mathcal{A}$ and the free names of the underlying process), a restricted process name can be revealed as x, and then $\mathcal{A}$ can be satisfied".

These two definitions will turn out to be equivalent (Logical Corollaries 5-4). The equivalence of the existential and universal definitions is at the essence of the notion of "freshness" [9]. Before coming back to $(\nu x)\mathcal{A}$, we study freshness in the next section, independently of revelation.

4 Fresh-Name Quantifier

In this section we define a formula, $\text{И}x.\mathcal{A}$, with the meaning "for fresh x, $\mathcal{A}$ holds". Here, "fresh" means, informally, distinct from any name that might clash with an existing name.

The set of free (i.e., non-fresh) names that occur in a process or formula is always finite; hence sets of fresh names are always cofinite[2]. If there is a suitable fresh x, then there are infinitely many of them, since a fresh name can be replaced by any other fresh name. Therefore, "freshness" can be expressed formally as the existence of a cofinite set of interchangeable names [9].

We use $Fin(S)$ for the collection of finite subsets of a set S, and (rarely) $CoFin(S)$ for the collection of cofinite subsets of S.

4.1 The Gabbay-Pitts Property

We would like to obtain the following property for $\text{И}x.\mathcal{A}$:

[2]. A cofinite set is the complement of a finite set with respect to an infinite universe, which, in our case, is the countable universe of names Λ.

$$P \vDash \text{И}x.\mathcal{A} \quad\Leftrightarrow\quad \exists m\in\Lambda.\ m\notin fn(P,\mathcal{A}) \wedge P \vDash \mathcal{A}\{x\leftarrow m\}$$

That is, $P \vDash \text{И}x.\mathcal{A}$ iff there exists a fresh name m such that $P \vDash \mathcal{A}\{x\leftarrow m\}$.

This definition is given by existential quantification over fresh names. Remarkably, there is an equivalent definition based on universal quantification. The equivalence of these two definitions is based on a deep property of the logic (Lemma 2-4), and will be used to great effect later. We state the equivalence as follows: there exists a fresh name m such that $P \vDash \mathcal{A}\{x\leftarrow m\}$, if and only if for all fresh names m we have $P \vDash \mathcal{A}\{x\leftarrow m\}$:

4-1 Proposition (Gabbay-Pitts Property)

$\forall P\in\Pi, \mathcal{A}\in\Phi, N\in Fin(\Lambda).$
$N \supseteq fn(P,\mathcal{A}) \wedge fv(\mathcal{A}) \subseteq \{x\} \Rightarrow$
$(\exists m\in\Lambda.\ m\notin N \wedge P \vDash \mathcal{A}\{x\leftarrow m\}) \;\Leftrightarrow\; (\forall m\in\Lambda.\ m\notin N \Rightarrow P \vDash \mathcal{A}\{x\leftarrow m\})$

Proof

Assume $N \supseteq fn(P,\mathcal{A})$ and $fv(\mathcal{A}) \subseteq \{x\}$.

Case $\Leftarrow$) Assume $\forall m\in\Lambda.\ m\notin N \Rightarrow P \vDash \mathcal{A}\{x\leftarrow m\}$. Since N is finite and Λ is infinite, there is a $p\in\Lambda$ such that $p\notin N$. Then, by assumption, $P \vDash \mathcal{A}\{x\leftarrow p\}$. We have shown $(\exists p\in\Lambda.\ p\notin N \wedge P \vDash \mathcal{A}\{x\leftarrow p\})$.

Case $\Rightarrow$) Assume $\exists m\in\Lambda.\ m\notin N \wedge P \vDash \mathcal{A}\{x\leftarrow m\}$; in particular, $m\notin fn(P,\mathcal{A})$. Take any $p\in\Lambda$ and assume $p\notin N$. If $p=m$ we have by assumption that $P \vDash \mathcal{A}\{x\leftarrow p\}$. Otherwise, if $p\neq m$ then $p\notin N\cup\{m\}$; since $fn(P,\mathcal{A}\{x\leftarrow m\}) \subseteq N\cup\{m\}$, we have that $p\notin fn(P,\mathcal{A}\{x\leftarrow m\})$. By applying Lemma 2-4 to the assumption $P \vDash \mathcal{A}\{x\leftarrow m\}$ we obtain $P\{m\leftarrow p\} \vDash \mathcal{A}\{x\leftarrow m\}\{m\leftarrow p\}$; that is, $P \vDash \mathcal{A}\{x\leftarrow p\}$. In both cases, we have shown that $(\forall p\in\Lambda.\ p\notin N \Rightarrow P \vDash \mathcal{A}\{x\leftarrow p\})$.

$\square$

The following corollary gives consequences and alternative formulations of the Gabbay-Pitts property in terms of finite sets of names that include $fn(P,\mathcal{A})$ (or, alternatively, in terms of cofinite sets of names that do not intersect $fn(P,\mathcal{A})$).

4-2 Corollary

Assume $P\in\Pi, \mathcal{A}\in\Phi, fv(\mathcal{A}) \subseteq \{x\}$. Then,
$\exists m\in\Lambda.\ m\notin fn(P,\mathcal{A}) \wedge P \vDash \mathcal{A}\{x\leftarrow m\}$

(1) $\Leftrightarrow\ \exists N\in Fin(\Lambda).\ N \supseteq fn(P,\mathcal{A}) \wedge \exists m\in\Lambda.\ m\notin N \wedge P \vDash \mathcal{A}\{x\leftarrow m\}$
(2) $\Leftrightarrow\ \exists N\in Fin(\Lambda).\ N \supseteq fn(P,\mathcal{A}) \wedge \forall m\in\Lambda.\ m\notin N \Rightarrow P \vDash \mathcal{A}\{x\leftarrow m\}$
(3) $\Leftrightarrow\ \forall N\in Fin(\Lambda).\ N \supseteq fn(P,\mathcal{A}) \Rightarrow \exists m\in\Lambda.\ m\notin N \wedge P \vDash \mathcal{A}\{x\leftarrow m\}$
(4) $\Leftrightarrow\ \forall N\in Fin(\Lambda).\ N \supseteq fn(P,\mathcal{A}) \Rightarrow \forall m\in\Lambda.\ m\notin N \Rightarrow P \vDash \mathcal{A}\{x\leftarrow m\}$

Proof

(1)

Case $\Rightarrow$) Assume $\exists m\in\Lambda.\ m\notin fn(P,\mathcal{A}) \wedge P \vDash \mathcal{A}\{x\leftarrow m\}$. Take $N = fn(P,\mathcal{A})$ to obtain $\exists N\in Fin(\Lambda).\ N \supseteq fn(P,\mathcal{A}) \wedge \exists m\in\Lambda.\ m\notin N \wedge P \vDash \mathcal{A}\{x\leftarrow m\}$.

Case $\Leftarrow$) Assume $\exists N{\in}Fin(\Lambda).\ N \supseteq fn(P,\mathcal{A}) \wedge \exists m{\in}\Lambda.\ m{\notin}N \wedge P \vDash \mathcal{A}\{x{\leftarrow}m\}$. In particular, $\exists m{\in}\Lambda.\ m{\notin}\, fn(P,\mathcal{A}) \wedge P \vDash \mathcal{A}\{x{\leftarrow}m\}$.

(2) Assume $\exists m{\in}\Lambda.\ m{\notin}\, fn(P,\mathcal{A}) \wedge P \vDash \mathcal{A}\{x{\leftarrow}m\}$. By (1), this is equivalent to $\exists N{\in}Fin(\Lambda).\ N \supseteq fn(P,\mathcal{A}) \wedge \exists m{\in}\Lambda.\ m{\notin}N \wedge P \vDash \mathcal{A}\{x{\leftarrow}m\}$. By Proposition 4-1, this is in turn equivalent to $\exists N{\in}Fin(\Lambda).\ N \supseteq fn(P,\mathcal{A}) \wedge \forall m{\in}\Lambda.\ m{\notin}N \Rightarrow P \vDash \mathcal{A}\{x{\leftarrow}m\}$.

(3)

Case $\Rightarrow$) Assume $\exists m{\in}\Lambda.\ m{\notin}fn(P,\mathcal{A}) \wedge P \vDash \mathcal{A}\{x{\leftarrow}m\}$. Take any $N \supseteq fn(P,\mathcal{A})$. From the assumption, by Proposition 4-1, we have $\forall m{\in}\Lambda.\ m{\notin}fn(P,\mathcal{A}) \Rightarrow P \vDash \mathcal{A}\{x{\leftarrow}m\}$. In particular, $\forall m{\in}\Lambda.\ m{\notin}N \Rightarrow P \vDash \mathcal{A}\{x{\leftarrow}m\}$. Then, by Proposition 4-1, $\exists m{\in}\Lambda.\ m{\notin}N \wedge P \vDash \mathcal{A}\{x{\leftarrow}m\}$. We have shown $\forall N{\in}Fin(\Lambda).\ N \supseteq fn(P,\mathcal{A}) \Rightarrow \exists m{\in}\Lambda.\ m{\notin}N \wedge P \vDash \mathcal{A}\{x{\leftarrow}m\}$.

Case $\Leftarrow$) Assume $\forall N{\in}Fin(\Lambda).\ N \supseteq fn(P,\mathcal{A}) \Rightarrow \exists m{\in}\Lambda.\ m{\notin}N \wedge P \vDash \mathcal{A}\{x{\leftarrow}m\}$. Take $N = fn(P,\mathcal{A})$ to obtain $\exists N{\in}Fin(\Lambda).\ N \supseteq fn(P,\mathcal{A}) \wedge \exists m{\in}\Lambda.\ m{\notin}N \wedge P \vDash \mathcal{A}\{x{\leftarrow}m\}$. By (1), $\exists m{\in}\Lambda.\ m{\notin}fn(P,\mathcal{A}) \wedge P \vDash \mathcal{A}\{x{\leftarrow}m\}$.

(4)

Case $\Rightarrow$) Assume $\exists m{\in}\Lambda.\ m{\notin}fn(P,\mathcal{A}) \wedge P \vDash \mathcal{A}\{x{\leftarrow}m\}$. By (3), $\forall N{\in}Fin(\Lambda).\ N \supseteq fn(P,\mathcal{A}) \Rightarrow \exists m{\in}\Lambda.\ m{\notin}N \wedge P \vDash \mathcal{A}\{x{\leftarrow}m\}$. Take any $N{\in}Fin(\Lambda)$ such that $N \supseteq fn(P,\mathcal{A})$; by assumption $\exists m{\in}\Lambda.\ m{\notin}N \wedge P \vDash \mathcal{A}\{x{\leftarrow}m\}$, and by Proposition 4-1, $\forall m{\in}\Lambda.\ m{\notin}N \Rightarrow P \vDash \mathcal{A}\{x{\leftarrow}m\}$. We have shown $\forall N{\in}Fin(\Lambda).\ N \supseteq fn(P,\mathcal{A}) \Rightarrow \forall m{\in}\Lambda.\ m{\notin}N \Rightarrow P \vDash \mathcal{A}\{x{\leftarrow}m\}$.

Case $\Leftarrow$) Assume $\forall N{\in}Fin(\Lambda).\ N \supseteq fn(P,\mathcal{A}) \Rightarrow \forall m{\in}\Lambda.\ m{\notin}N \Rightarrow P \vDash \mathcal{A}\{x{\leftarrow}m\}$. Take $N = fn(P,\mathcal{A})$ to obtain $\exists N{\in}Fin(\Lambda).\ N \supseteq fn(P,\mathcal{A}) \wedge \forall m{\in}\Lambda.\ m{\notin}N \Rightarrow P \vDash \mathcal{A}\{x{\leftarrow}m\}$. By (2), $\exists m{\in}\Lambda.\ m{\notin}fn(P,\mathcal{A}) \wedge P \vDash \mathcal{A}\{x{\leftarrow}m\}$.

□

4.2 A Gabbay-Pitts Logical Rule

We now want to formulate a Gabbay-Pitts property similar to Proposition 4-1, but expressible within the logic. We are going to use extensively the idiom $x\#N \wedge x®\mathbf{T}$, for a quantified variable x. The first part of this conjunction says that the name x is fresh with respect to a given set of names N that usually includes the set of free names of a formula of interest. The second part says that x is fresh in the "underlying process", because $P \vDash n®\mathbf{T}$ iff $n{\notin}fn(P)$. For a suitable choice of N, the whole conjunction can be understood as saying that x is "completely fresh", both at the formula and process level, in a given situation.

Notation

- For $N{\in}Fin(\Lambda{\cup}\vartheta)$ we define the formula $\eta\#N \triangleq \bigwedge_{\mu\in N}(\eta{\neq}\mu)$.
  For any P and closed $m\#N$, we have $P \vDash m\#N$ iff $m{\notin}N$.
- Let $fnv(\mathcal{A}) \triangleq fn(\mathcal{A}){\cup}fv(\mathcal{A})$, so that $fnv(\mathcal{A}){\in}Fin(\Lambda{\cup}\vartheta)$

With this understanding, the following proposition states the single rule (schema) that we add to our logic in order to capture "freshness", and establishes its soundness. Note that this rule holds for open formulas.

4-3 Proposition (Validity: Gabbay-Pitts)

(GP)$\}$ $\exists x.\ x\#N \wedge x®\mathbf{T} \wedge \mathcal{A} \dashv\vdash \forall x.\ (x\#N \wedge x®\mathbf{T}) \Rightarrow \mathcal{A}$
where $N\in Fin(\Lambda\cup\vartheta)$ and $N \supseteq fnv(\mathcal{A})\text{-}\{x\}$ and $x\notin N$

Proof

Assume $N \supseteq fnv(\mathcal{A})\text{-}\{x\}$ and $x\notin N$. We need to show that the sequent is valid, that is that $\forall\varphi\in(fv(\mathcal{A})\text{-}\{x\})\rightarrow\Lambda,\ P\in\Pi.\ P \vDash (\exists x.\ x\#N \wedge x®\mathbf{T} \wedge \mathcal{A})_\varphi \Leftrightarrow P \vDash (\forall x.\ x\#N \wedge x®\mathbf{T} \Rightarrow \mathcal{A})_\varphi$.

(1) $\}$ $\exists x.\ x\#N \wedge x®\mathbf{T} \wedge \mathcal{A} \vdash \forall x.\ x\#N \wedge x®\mathbf{T} \Rightarrow \mathcal{A}$

Take any $\varphi\in(fv(\mathcal{A})\text{-}\{x\})\rightarrow\Lambda$ and $P\in\Pi$, and assume $P \vDash (\exists x.\ x\#N \wedge x®\mathbf{T} \wedge \mathcal{A})_\varphi$. That is, assume $\exists m\in\Lambda.\ m\notin N_\varphi\cup fn(P) \wedge P \vDash \mathcal{A}_\varphi\{x\leftarrow m\}$, where $N_\varphi\cup fn(P)\supseteq fn(P,\mathcal{A}_\varphi)$ and $fv(\mathcal{A}_\varphi) \subseteq \{x\}$. By Proposition 4-1, we obtain $\forall m\in\Lambda.\ m\notin N_\varphi\cup fn(P) \Rightarrow P \vDash \mathcal{A}_\varphi\{x\leftarrow m\}$, that is $P \vDash (\forall x.\ x\#N \wedge x®\mathbf{T} \Rightarrow \mathcal{A})_\varphi$.

(2) $\}$ $\forall x.\ x\#N \wedge x®\mathbf{T} \Rightarrow \mathcal{A} \vdash \exists x.\ x\#N \wedge x®\mathbf{T} \wedge \mathcal{A}$

Take any $\varphi\in(fv(\mathcal{A})\text{-}\{x\})\rightarrow\Lambda$ and $P\in\Pi$ and assume $P \vDash (\forall x.\ x\#N \wedge x®\mathbf{T} \Rightarrow \mathcal{A})_\varphi$; that is assume $(\forall m\in\Lambda.\ m\notin N_\varphi\cup fn(P) \Rightarrow P \vDash \mathcal{A}_\varphi\{x\leftarrow m\})$, where $N_\varphi\cup fn(P)\supseteq fn(P,\mathcal{A}_\varphi)$ and $fv(\mathcal{A}_\varphi) \subseteq \{x\}$. By Proposition 4-1, we obtain $\exists m\in\Lambda.\ m\notin N_\varphi\cup fn(P) \wedge P \vDash \mathcal{A}_\varphi\{x\leftarrow m\}$, that is $P \vDash (\exists x.\ x\#N \wedge x®\mathbf{T} \wedge \mathcal{A})_\varphi$.

□

Remark. (GP) gives us a way to prove that $\forall x.\mathcal{A} \vdash \exists x.\mathcal{A}$. This depends on the fact that the set of names is non-empty, and is obviously not derivable from the normal quantifier rules. Take $N = fnv(\mathcal{A})\text{-}\{x\}$. Starting from $\mathcal{A} \vdash \mathcal{A}$, by right weakening and quantifier introduction we obtain $\forall x.\ \mathcal{A} \vdash \forall x.\ x\#N \wedge x®\mathbf{T} \Rightarrow \mathcal{A}$. Again starting from $\mathcal{A} \vdash \mathcal{A}$, by left weakening and quantifier introduction we obtain $\exists x.\ x\#N \wedge x®\mathbf{T} \wedge \mathcal{A} \vdash \exists x.\ \mathcal{A}$. By (GP) we have $\forall x.\ x\#N \wedge x®\mathbf{T} \Rightarrow \mathcal{A} \vdash \exists x.\ x\#N \wedge x®\mathbf{T} \wedge \mathcal{A}$. Hence, by transitivity we obtain $\forall x.\ \mathcal{A} \vdash \exists x.\ \mathcal{A}$. □

4.3 Fresh-Name Quantifier

Now, we can define quantification over fresh names, $\text{И}x.\mathcal{A}$, as follows:

4-4 Definition (Fresh-Name Quantifier)

$\text{И}x.\mathcal{A} \ \triangleq\ \exists x.\ x\#fnv(\mathcal{A})\text{-}\{x\} \wedge x®\mathbf{T} \wedge \mathcal{A}$

□

Hence $fn(\text{И}x.\mathcal{A}) = fn(\mathcal{A})$ and $fv(\text{И}x.\mathcal{A}) = fv(\mathcal{A})\text{-}\{x\}$.

Note that the right-hand side of this definition depends on the set of free names and variables of $\mathcal{A}$. Therefore, this is not a definition within the logic, but rather a meta-theoretical definition (or abbreviation) that should always be understood in its expanded form. Any general theorem or derived rule involving $\text{И}x.\mathcal{A}$ will in fact be a schematic theorem or rule with respect to the free names and variables of $\mathcal{A}$, in the same way that (GP) is a rule schema.

By (GP) (Proposition 4-3) we have:

$$\text{И}x.\mathcal{A} \dashv\vdash \forall x.\ x\#fnv(\mathcal{A})\text{-}\{x\} \wedge x\circledR\mathbf{T} \Rightarrow \mathcal{A}$$

In terms of satisfaction, we obtain:

4-5 Lemma ($P \vDash \text{И}x.\mathcal{A}$)

$P \vDash \text{И}x.\mathcal{A}$
 iff $\exists m\in\Lambda.\ m\notin fn(P,\mathcal{A}) \wedge P \vDash \mathcal{A}\{x\leftarrow m\}$
 iff $\forall m\in\Lambda.\ m\notin fn(P,\mathcal{A}) \Rightarrow P \vDash \mathcal{A}\{x\leftarrow m\}$

Proof

Note that $P \vDash \text{И}x.\mathcal{A}$ implies that $fv(\text{И}x.\mathcal{A}) = \phi$, that is $fv(\mathcal{A}) \subseteq \{x\}$. Similarly, $P \vDash \mathcal{A}\{x\leftarrow m\}$ implies that $fv(\mathcal{A}\{x\leftarrow m\}) = \phi$, that is $fv(\mathcal{A}) \subseteq \{x\}$.

By definition of И, $P \vDash \text{И}x.\mathcal{A}$ iff $P \vDash \exists x.\ x\#fnv(\mathcal{A})\text{-}\{x\} \wedge x\circledR\mathbf{T} \wedge \mathcal{A}$. Since $fv(\mathcal{A}) \subseteq \{x\}$, this is equivalent to $P \vDash \exists x.\ x\#fn(\mathcal{A}) \wedge x\circledR\mathbf{T} \wedge \mathcal{A}$; that is $\exists m\in\Lambda.\ P \vDash m\#fn(\mathcal{A}) \wedge P \vDash m\circledR\mathbf{T} \wedge P \vDash \mathcal{A}\{x\leftarrow m\}$. This is the same as $\exists m\in\Lambda.\ m\notin fn(\mathcal{A}) \wedge m\notin fn(P) \wedge P \vDash \mathcal{A}\{x\leftarrow m\}$.

The second equivalence is obtained by Proposition 4-1.

□

Therefore, $\text{И}x.\mathcal{A}$ can be understood as saying either that there is a fresh name x such that $\mathcal{A}$ holds, or that for any fresh name x we have that $\mathcal{A}$ holds. These formulations are equivalent because of the cofinite nature of sets of fresh names. If there is a suitably fresh x such that $\mathcal{A}$ holds, then any other fresh name will work equally well, so all fresh names will work. Conversely, if for all suitably fresh names $\mathcal{A}$ holds, since any set of fresh names is (cofinite and hence) non-empty, there exists a fresh name for which $\mathcal{A}$ holds.

Remark. The meaning of $\text{И}x.\mathcal{A}$ when $\mathcal{A}$ has free variables other than x is subtle. When we write $\text{И}x.\ ...n...$ we intend x to be fresh w.r.t. any existing name, and in particular n; similarly, when we write $\text{И}x.\ ...y...$ we intend x to be fresh with respect to any name denoted by y. Consider $\text{И}y.\ \text{И}x.\ y{=}x$; this formula should not be valid. In fact, it is contradictory because, by definition, it means, $\exists y.\ y\circledR\mathbf{T} \wedge \exists x.\ x{\neq}y \wedge x\circledR\mathbf{T} \wedge y{=}x$. Similarly, $\forall y.\ \text{И}x.\ y{=}x$ and $\exists y.\ \text{И}x.\ y{=}x$ are contradictory. (Instead, $\text{И}x.\ \exists y.\ x{=}y$ is valid.) □

The following rules are now derivable entirely within the logic:

4-6 Logical Corollaries (Fresh-Name Quantifier)

(И ∃)	$\}\ \text{И}x.\mathcal{A} \dashv\vdash \exists x.\ x\#N \wedge x\circledR\mathbf{T} \wedge \mathcal{A}$	where $N \supseteq fnv(\mathcal{A})\text{-}\{x\}$ and $x\notin N$
(И ∀)	$\}\ \forall x.\ x\#N \wedge x\circledR\mathbf{T} \Rightarrow \mathcal{A} \dashv\vdash \text{И}x.\mathcal{A}$	where $N \supseteq fnv(\mathcal{A})\text{-}\{x\}$ and $x\notin N$
(И ¬)	$\}\ \neg\text{И}x.\mathcal{A} \dashv\vdash \text{И}x.\neg\mathcal{A}$	
(И |)	$\}\ \text{И}x.(\mathcal{A} \mid \mathcal{B}) \dashv\vdash (\text{И}x.\mathcal{A}) \mid (\text{И}x.\mathcal{B})$	
(И ⊢)	$\mathcal{A} \vdash \mathcal{B}\ \}\ \text{И}x.\mathcal{A} \vdash \text{И}x.\mathcal{B}$	
(И fv)	$\}\ \text{И}x.\mathcal{A} \dashv\vdash \mathcal{A}$	where $x\notin fv(\mathcal{A})$
(И n[])	$\}\ \text{И}x.y[\mathcal{A}] \dashv\vdash y[\text{И}x.\mathcal{A}]$	where $x{\neq}y$
(И R)	$\mathcal{A} \wedge x\#N \wedge x\circledR\mathbf{T} \vdash \mathcal{B}\ \}\ \mathcal{A} \vdash \text{И}x.\mathcal{B}$	where $N \supseteq fnv(\mathcal{B})\text{-}\{x\}$ and $x \notin N \cup fv(\mathcal{A})$

(И L) $\mathcal{A} \wedge x\#N \wedge x®\mathbf{T} \vdash \mathcal{B} \;\}\; \text{И}x.\mathcal{A} \vdash \mathcal{B}$ where $N \supseteq fnv(\mathcal{A})\text{-}\{x\}$ and $x \notin N \cup fv(\mathcal{B})$
(И E) $\mathcal{A} \vdash \text{И}x.\mathcal{B};\ \mathcal{B} \wedge x\#N \wedge x®\mathbf{T} \vdash \mathcal{C} \;\}\; \mathcal{A} \vdash \mathcal{C}$ where $N \supseteq fnv(\mathcal{B})\text{-}\{x\}$ and $x \notin N \cup fv(\mathcal{C})$
□

Remark. The fresh-name quantifier is "in between" universal and existential quantification ($\forall x.\mathcal{A} \vdash \text{И}x.\mathcal{A} \vdash \exists x.\mathcal{A}$, by (И ∀) and (И ∃)). Moreover, it is "right in the middle", since it enjoys many properties of both universal and existential quantification; for example, it is self-dual ($\text{И}x.\mathcal{A} \dashv\vdash \neg\text{И}x.\neg\mathcal{A}$, by (И ∀), (И ∃), and DeMorgan). □

Remark. We can show that $\forall y.\ \exists x.\ x{\neq}y$. Start from $x{\neq}y \wedge x®\mathbf{T} \Rightarrow x{\neq}y$, and generalize to $\forall y.\ \forall x.\ x{\neq}y \wedge x®\mathbf{T} \Rightarrow x{\neq}y$. By (И ∀) this means $\forall y.\ \text{И}x.\ x{\neq}y$. By the previous remark, $\text{И}x.\ x{\neq}y \vdash \exists x.\ x{\neq}y$, hence by ($\forall \vdash$) we obtain $\forall y.\ \exists x.\ x{\neq}y$. □

Remark. Of particular interest (and difficulty) is the distribution of И over |, rule (И |):

$$\}\ \text{И}x.(\mathcal{A} \mid \mathcal{B}) \dashv\vdash (\text{И}x.\mathcal{A}) \mid (\text{И}x.\mathcal{B})$$

Distribution over | holds in one direction for universal quantification, in the other direction for existential quantification, and in both directions for fresh-name quantification. This rule can be understood informally as follows (this is a sketch of the formal derivation). In the left-to-right direction we use the existential interpretation of И. Take any P; if $P \vDash \text{И}x.(\mathcal{A} \mid \mathcal{B})$ then there are a fresh name x and processes P', P'' such that $P \equiv P' \mid P''$ and $P' \vDash \mathcal{A}$ and $P'' \vDash \mathcal{B}$. Hence, there is a fresh name x such that $P' \vDash \mathcal{A}$ and again a fresh name x such that $P'' \vDash \mathcal{B}$; that is, $P' \vDash \text{И}x.\mathcal{A}$ and $P'' \vDash \text{И}x.\mathcal{B}$. Therefore, $P \equiv P' \mid P'' \vDash (\text{И}x.\mathcal{A}) \mid (\text{И}x.\mathcal{B})$. In the right-to-left direction we use the universal interpretation of И. Take any P; if $P \vDash (\text{И}x.\mathcal{A}) \mid (\text{И}x.\mathcal{B})$ then there are processes P', P'' such that $P \equiv P' \mid P''$ and $P' \vDash \text{И}x.\mathcal{A}$ and $P'' \vDash \text{И}x.\mathcal{B}$. This means that for all names x' fresh in P' and $\mathcal{A}$, we have $P' \vDash \mathcal{A}\{x{\leftarrow}x'\}$ and for all names x'' fresh in P'' and $\mathcal{B}$, we have $P'' \vDash \mathcal{B}\{x{\leftarrow}x''\}$. Now, for all names y that are fresh in P', $\mathcal{A}$, P'', $\mathcal{B}$; we have that $P' \vDash \mathcal{A}\{x{\leftarrow}y\}$ and $P'' \vDash \mathcal{B}\{x{\leftarrow}y\}$. That is, $P \equiv P' \mid P'' \vDash \text{И}y.(\mathcal{A}\{x{\leftarrow}y\} \mid \mathcal{B}\{x{\leftarrow}y\}) = \text{И}x.(\mathcal{A} \mid \mathcal{B})$. □

5 Hidden-Name Quantifier

5.1 Definition

We can finally get back to our original aim of defining a hidden-name quantifier at the logical level. We take:

5-1 Definition (Hidden-Name Quantifier)

$$(\nu x)\mathcal{A} \triangleq \text{И}x.x®\mathcal{A}$$

□

Hence $fn((\nu x)\mathcal{A}) = fn(\mathcal{A})$ and $fv((\nu x)\mathcal{A}) = fv(\mathcal{A})\text{-}\{x\}$. Moreover, by definition of И:

$$(\nu x)\mathcal{A} \;=\; \exists x.\ x\#fnv(\mathcal{A})\text{-}\{x\} \wedge x®\mathbf{T} \wedge x®\mathcal{A}$$

and, because of Logical Corollary 3-3(® ∧), we can simplify this to:

$$(\nu x)\mathcal{A} \;\dashv\vdash\; \exists x.\ x\#fnv(\mathcal{A})\text{-}\{x\} \wedge x®\mathcal{A}$$

In terms of satisfaction, we obtain:

5-2 Lemma ($P \vDash (\nu x)\mathcal{A}$)

$P \vDash (\nu x)\mathcal{A}$ iff

$\exists m \in \Lambda.\ m \notin fn(P,\mathcal{A}) \wedge \exists P' \in \Pi.\ P \equiv (\nu m)P' \wedge P' \vDash \mathcal{A}\{x \leftarrow m\}$

Proof

Satisfaction is defined for closed formulas, so $\exists x.\ x\#fnv(\mathcal{A})\text{-}\{x\} \wedge x\circledR\mathbf{T} \wedge x\circledR\mathcal{A}$ is the same as $\exists x.\ x\#fn(\mathcal{A}) \wedge x\circledR\mathbf{T} \wedge x\circledR\mathcal{A}$. Then, $P \vDash \exists x.\ x\#fn(\mathcal{A}) \wedge x\circledR\mathbf{T} \wedge x\circledR\mathcal{A}$ means that $\exists m \in \Lambda.\ m \notin fn(\mathcal{A}) \wedge m \notin fn(P) \wedge P \vDash m\circledR\mathcal{A}\{x \leftarrow m\}$, and from definition of satisfaction for $\circledR$, we obtain the statement.

□

5.2 Properties

We can now verify that this definition of the hidden-name quantifier fully satisfies the property (νx-proper):

5-3 Proposition (νx-proper)

For all $n \in \Lambda$, $x \in \vartheta$, $P \in \Pi$, and closed $\mathcal{A} \in \Phi$:

$n \notin fn(P) \wedge P \vDash (\nu x)(\mathcal{A}\{n \leftarrow x\}) \quad \Leftrightarrow \quad \exists P' \in \Pi.\ P \equiv (\nu n)P' \wedge P' \vDash \mathcal{A}$

Proof

Case $\Rightarrow$)

$P \vDash (\nu x)(\mathcal{A}\{n \leftarrow x\}) \Rightarrow P \vDash \text{И}x.x\circledR(\mathcal{A}\{n \leftarrow x\})$

$\Rightarrow \exists m \in \Lambda.\ m \notin fn(P,x\circledR(\mathcal{A}\{n \leftarrow x\})) \wedge \exists P' \in \Pi.\ P \equiv (\nu m)P' \wedge P' \vDash \mathcal{A}\{n \leftarrow x\}\{x \leftarrow m\}$

$\Rightarrow \exists m \in \Lambda.\ m \notin fn(P,\mathcal{A}\{n \leftarrow x\}) \wedge \exists P' \in \Pi.\ P \equiv (\nu m)P' \wedge P' \vDash \mathcal{A}\{n \leftarrow m\}$

Suppose $m = n$. Then we immediately obtain $\exists P' \in \Pi.\ P \equiv (\nu n)P' \wedge P' \vDash \mathcal{A}$.

Suppose $m \neq n$. Since $n \notin fn(P)$ by assumption and $P \equiv (\nu m)P'$ we have $n \notin fn(P')$ by Lemma 2-2(1). Take $P'' = P'\{m \leftarrow n\}$. By Lemma 2-4, since $n \notin fn(P', \mathcal{A}\{n \leftarrow m\})$ and $P' \vDash \mathcal{A}\{n \leftarrow m\}$, we obtain $P'\{m \leftarrow n\} \vDash \mathcal{A}\{n \leftarrow m\}\{m \leftarrow n\}$, that is, $P'' \vDash \mathcal{A}$. Now, $P \equiv (\nu m)P' = (\nu n)P'\{m \leftarrow n\} = (\nu n)P''$, so that $P \equiv (\nu n)P''$. We have shown that $\exists P'' \in \Pi.\ P \equiv (\nu n)P'' \wedge P'' \vDash \mathcal{A}$.

Case $\Leftarrow$)

Assume $\exists P' \in \Pi.\ P \equiv (\nu n)P' \wedge P' \vDash \mathcal{A}$; then $n \notin fn(P)$ by Lemma 2-2(1). Moreover:

$\exists P' \in \Pi.\ P \equiv (\nu n)P' \wedge P' \vDash \mathcal{A} \Rightarrow P \vDash n\circledR\mathcal{A}$

$\Rightarrow P \vDash (x\circledR\mathcal{A}\{n \leftarrow x\})\{x \leftarrow n\}$ (since $n\circledR\mathcal{A}$ is closed and hence $x \notin fv(\mathcal{A})$)

$\Rightarrow \exists n \in \Lambda.\ n \notin fn(P,\ x\circledR(\mathcal{A}\{n \leftarrow x\})) \wedge P \vDash (x\circledR\mathcal{A}\{n \leftarrow x\})\{x \leftarrow n\}$

$\Rightarrow P \vDash \text{И}x.x\circledR\mathcal{A}\{n \leftarrow x\} \Rightarrow P \vDash (\nu x)(\mathcal{A}\{n \leftarrow x\})$

□

Remark. We saw that for the existential definition of the hidden-name quantifier, we obtained the undesirable property $(\nu n)n[] \vDash \exists x.x\circledR p[]$. Since $n[] \nvDash p[]$, Proposition 5-3 implies that $(\nu n)n[] \nvDash \text{И}x.x\circledR p[]$. As a sanity check, suppose that $(\nu n)n[] \vDash \text{И}x.x\circledR p[]$; this would mean that $\exists m{\in}\Lambda.\ m{\notin}fn((\nu n)n[],p[]) \wedge \exists P'{\in}\Pi.\ (\nu n)n[] \equiv (\nu m)P' \wedge P' \vDash p[]$. But $P' \vDash p[]$ implies $P' \equiv p[]$, and since $m \neq p$ we cannot have $(\nu n)n[] \equiv (\nu m)P'$ by Lemma 2-2(1). □

Remark. We saw that for the universal definition of the hidden-name quantifier, we obtained the undesirable property $(\nu n)p[] \nvDash \forall x.x\circledR\neg x[]$. Since $p[] \vDash \neg n[]$, Proposition 5-3 implies that $(\nu n)p[] \vDash \text{И}x.x\circledR\neg x[]$. In fact, $p[] \vDash \neg n[]$ implies $(\nu n)p[] \vDash n\circledR\neg n[]$, which is the same as $(\nu n)p[] \vDash (x\circledR\neg x[])\{x{\leftarrow}n\}$, where $n{\notin}fn((\nu n)p[],x\circledR\neg x[])$. Therefore, $\exists m{\in}\Lambda.\ m{\notin}fn((\nu n)p[],x\circledR\neg x[]) \wedge (\nu n)p[] \vDash (x\circledR\neg x[])\{x{\leftarrow}m\}$, which means $(\nu n)p[] \vDash \text{И}x.x\circledR\neg x[]$ by Lemma 4-5. □

The following rules for $(\nu x)\mathcal{A}$ can be derived by combining the rules for revelation and for the fresh-name quantifier.

5-4 Logical Corollaries (Hidden-Name Quantifier)

$(\nu\ \forall)$	$\}\ \forall x.\ (x\#N \wedge x\circledR\mathbf{T}) \Rightarrow x\circledR\mathcal{A} \dashv\vdash (\nu x)\mathcal{A}$	where $N \supseteq fnv(\mathcal{A})\text{-}\{x\}$ and $x \notin N$
$(\nu\ \exists)$	$\}\ (\nu x)\mathcal{A} \dashv\vdash \exists x.\ x\#N \wedge x\circledR\mathbf{T} \wedge x\circledR\mathcal{A}$ $\dashv\vdash \exists x.\ x\#N \wedge x\circledR\mathcal{A}$	where $N \supseteq fnv(\mathcal{A})\text{-}\{x\}$ and $x \notin N$
$(\nu\ \mathrm{R})$	$\mathcal{A} \wedge x\#N \wedge x\circledR\mathbf{T} \vdash x\circledR\mathcal{B}\ \}\ \mathcal{A} \vdash (\nu x)\mathcal{B}$	where $N \supseteq fnv(\mathcal{B})\text{-}\{x\}$ and $x \notin N \cup fv(\mathcal{A})$
$(\nu\ \mathrm{L})$	$x\circledR\mathcal{A} \wedge x\#N \vdash \mathcal{B}\ \}\ (\nu x)\mathcal{A} \vdash \mathcal{B}$	where $N \supseteq fnv(\mathcal{A})\text{-}\{x\}$ and $x \notin N \cup fv(\mathcal{B})$
$(\nu\ \mathrm{E})$	$\mathcal{A} \vdash (\nu x)\mathcal{B};\ x\circledR\mathcal{B} \wedge x\#N \vdash \mathcal{C}\ \}\ \mathcal{A} \vdash \mathcal{C}$	where $N \supseteq fnv(\mathcal{B})\text{-}\{x\}$ and $x \notin N \cup fv(\mathcal{C})$
$(\nu \vdash)$	$\mathcal{A} \vdash \mathcal{B}\ \}\ (\nu x)\mathcal{A} \vdash (\nu x)\mathcal{B}$	
$(\nu\, fv)$	$\}\ (\nu x)(\mathcal{A}\oslash x) \dashv\vdash \mathcal{A}$	where $x \notin fv(\mathcal{A})$
$(\nu\, fv)$	$\}\ \mathcal{A} \vdash (\nu x)\mathcal{A}$	where $x \notin fv(\mathcal{A})$
$(\nu\ \circledR)$	$\}\ (\nu x)(x\circledR\mathcal{A}) \dashv\vdash (\nu x)\mathcal{A}$	
$(\nu\ \oslash)$	$\}\ (\nu x)(\mathcal{A}\oslash x) \vdash (\nu x)\mathcal{A}$	
$(\nu\ \mathbf{0})$	$\}\ (\nu x)\mathbf{0} \dashv\vdash \mathbf{0}$	
$(\nu\ n[])$	$\}\ (\nu x)y[\mathcal{A}] \dashv\vdash y[(\nu x)\mathcal{A}]$	where $x{\neq}y$
$(\nu \mid)$	$\}\ (\nu x)(\mathcal{A} \mid x\circledR\mathcal{B}) \dashv\vdash ((\nu x)\mathcal{A}) \mid ((\nu x)\mathcal{B})$	
$(\nu \oslash \mid)$	$\}\ (\nu x)(\mathcal{A}\oslash x) \mid (\nu x)(\mathcal{B}\oslash x) \dashv\vdash (\nu x)((\mathcal{A} \mid \mathcal{B})\oslash x)$	

□

Remark. We obtain $\forall x.\ x\circledR\mathcal{A} \vdash (\nu x)\mathcal{A} \vdash \exists x.\ x\circledR\mathcal{A}$. However, there are no interesting rules for $\neg(\nu x)\mathcal{A}$. □

Remark. This fails:

$\}\ (\nu x)\mathcal{A} \vdash \mathcal{A}$ where $x{\notin}fv(\mathcal{A})$

because $(\nu n)(n[] \mid n[]) \vDash \text{И}x.x\circledR(\neg\mathbf{0} \mid \neg\mathbf{0})$ but $(\nu n)(n[] \mid n[]) \nvDash \neg\mathbf{0} \mid \neg\mathbf{0}$. This is $\circledR$'s fault, not И's: $n\circledR\mathcal{A} \vdash \mathcal{A}$ fails with the same counterexample. □

Remark.

Property (νx-proper), Proposition 5-3, is derivable within the logic. That is:

$$n®\mathbf{T} \wedge (\nu x)(\mathcal{A}\{n\leftarrow x\}) \dashv\vdash n®\mathcal{A} \qquad \text{where } x \notin fv(\mathcal{A}):$$

In the left-to-right direction, from $n®\mathbf{T} \wedge (\nu x)(\mathcal{A}\{n\leftarrow x\})$ we obtain by definition $n®\mathbf{T} \wedge И x.x®\mathcal{A}\{n\leftarrow x\}$, and by (И ∀) we obtain $n®\mathbf{T} \wedge \forall x.\ x\#fnv(x®\mathcal{A}\{n\leftarrow x\})\text{-}\{x\} \wedge x®\mathbf{T} \Rightarrow x®\mathcal{A}\{n\leftarrow x\}$, which since $x \notin fv(\mathcal{A})$ is the same as $n®\mathbf{T} \wedge \forall x.\ x\#fnv(\mathcal{A})\text{-}\{n\} \wedge x®\mathbf{T} \Rightarrow x®\mathcal{A}\{n\leftarrow x\}$. By instantiating x to n, we get $n®\mathbf{T} \wedge (n\#fnv(\mathcal{A})\text{-}\{n\} \wedge n®\mathbf{T} \Rightarrow n®\mathcal{A}\{n\leftarrow n\})$. Since $n\#fnv(\mathcal{A})\text{-}\{n\}$ is true and is $n®\mathbf{T}$ assumed, we obtain $n®\mathcal{A}\{n\leftarrow n\}$, that is $n®\mathcal{A}$.
In the right-to-left direction, assuming $n®\mathcal{A}$, from $\mathcal{A} \vdash \mathbf{T}$ and (® ⊢) we obtain $n®\mathcal{A} \vdash n®\mathbf{T}$ and hence the first conjunct, $n®\mathbf{T}$. Then we trivially obtain $n\#fnv(n®\mathcal{A}\{n\leftarrow n\})\text{-}\{n\} \wedge n®\mathbf{T} \wedge n®\mathcal{A}\{n\leftarrow n\}$, since these conjuncts are either true or implied by $n®\mathcal{A}$. That formula is the same as $(x\#fnv(x®\mathcal{A}\{n\leftarrow x\})\text{-}\{x\} \wedge x®\mathbf{T} \wedge x®\mathcal{A}\{n\leftarrow x\})\{x\leftarrow n\}$ since $x \notin fv(\mathcal{A})$. Then, by (∃ R) we obtain $\exists x.\ x\#fnv(x®\mathcal{A}\{n\leftarrow x\})\text{-}\{x\} \wedge x®\mathbf{T} \wedge x®\mathcal{A}\{n\leftarrow x\}$. This implies, by (И ∃), that $И x.x®\mathcal{A}\{n\leftarrow x\}$, which is the same as $(\nu x)(\mathcal{A}\{n\leftarrow x\})$. □

5.3 Example

As an example of a specification containing a hidden-name quantifier, consider a situation where a secret is shared by two locations n and m, but is not known outside those locations.

We can state this as follows (recall that $©\eta \triangleq \neg\eta®\mathbf{T}$ and that $P \vDash ©n$ iff $n\in fn(P)$):

$$(\nu x)\ (n[©x] \mid m[©x])$$

It reads: for a fresh x, the name x is known at n and m, and is restricted anywhere else.

Expanding the definitions, we obtain:

$P \vDash (\nu x)\ (n[©x] \mid m[©x])$

$\Leftrightarrow P \vDash И x.x®(n[©x] \mid m[©x])$

$\Leftrightarrow \exists r\in\Lambda.\ r\notin fn(P)\cup\{n,m\} \wedge \exists Q\in\Pi.\ P \equiv (\nu r)Q \wedge Q \vDash (n[©x] \mid m[©x])\{x\leftarrow r\}$

$\Leftrightarrow \exists r\in\Lambda.\ r\notin fn(P)\cup\{n,m\} \wedge \exists Q\in\Pi.\ P \equiv (\nu r)Q \wedge Q \vDash n[©r] \mid m[©r]$

$\Leftrightarrow \exists r\in\Lambda.\ r\notin fn(P)\cup\{n,m\} \wedge \exists Q\in\Pi.\ P \equiv (\nu r)Q \wedge$
$\quad \exists Q',Q''\in\Pi.\ Q \equiv Q' \mid Q'' \wedge Q' \vDash n[©r] \wedge Q'' \vDash m[©r]$

$\Leftrightarrow \exists r\in\Lambda.\ r\notin fn(P)\cup\{n,m\} \wedge \exists Q\in\Pi.\ P \equiv (\nu r)Q \wedge$
$\quad \exists R',R''\in\Pi.\ Q \equiv n[R'] \mid m[R''] \wedge R' \vDash ©r \wedge R'' \vDash ©r$

$\Leftrightarrow \exists r\in\Lambda.\ r\notin fn(P)\cup\{n,m\} \wedge \exists R',R''\in\Pi.\ P \equiv (\nu r)(n[R'] \mid m[R'']) \wedge r\in fn(R') \wedge r\in fn(R'')$

The last line reads: P satisfies the specification iff there exists a name r that is fresh (not conflicting with n and m or public to P), such that r is known to the processes R' and R'' located at n and m, and is restricted inside P.

Here is a simple example of an implementation of this specification:

$$P = (\nu p)\ (n[p[]] \mid m[p[]])$$

6 Related Work and Conclusions

We have introduced a logic for describing concurrent processes with restricted names. Most previous logics for concurrency have strived to described properties that are invariant under some coarse process equivalence, such as bisimulation. Because of our original motivation in describing location structures in detail, the properties described by our logic are much finer, and are invariant only up to structural congruence (see also [12] for a recent characterization). Because of this, our logic is closely related to intuitionistic linear logic and to bunched logics: see [7] for a comparison. Our logic is unusual also because it handles variables ranging over a countable universe of names; these variables can be the subject of universal, existential, fresh-name, and hidden-name quantification.

Our logic is built directly out of a process model, so logical soundness is easy to check. Logical completeness is a much more difficult question. We do not expect the full logic to be complete with respect to our model (even for finite behaviors). Silvano dal Zilio is investigating some small, complete fragments of the logic. So far, we have mostly tried to discover as many true logical facts as possible (a measure of which is, for example, to be able to embed other logics into ours [7]), and to minimize the collection of basic rules. We have concentrated in particular on commutation and distribution properties of operators that can be useful in formal proofs.

In the present paper, fresh-name quantification is modeled after Gabbay and Pitts [9], adapted to our context; it provides logical rules for reasoning abstractly about freshness. Hidden-name quantification is obtained by combining fresh-name quantification with a revelation operator (not a quantifier) for revealing restricted process names. Most novel axioms have to do with revelation; they often reflect and resemble well-known properties of π-calculus restriction. Technically, we have added to our previous ambient logic just the revelation operator (and its adjunct) and an axiom schema expressing the Gabbay-Pitts property. In particular, fresh-name quantification, hidden-name quantification, and their properties, are derived.
Recently, we have become aware of related work by Luís Caires (both [3] and more recent unpublished work). Our aims are quite similar, but we are currently using different formal techniques; we are in the process of comparing results.

References

[1] Martín Abadi, **Secrecy by Typing in Security Protocols**. Journal of the ACM 46, 5 (September 1999), 749-786.

[2] Martín Abadi and Andrew D. Gordon, **A Calculus for Cryptographic Protocols: the Spi Calculus**. Information and Computation 148(1999):1-70.

[3] Luís Caires and Luís Monteiro, **Verifiable and Executable Logic Specifications of Concurrent Objects in $\mathcal{L}_\pi$.** In Programming Languages and Systems, Proceedings of ESOP'98, Chris Hankin (Ed.), Lecture Notes in Computer Science vol. 1877, Springer, 1998. pp. 42-56.

[4] Luca Cardelli, Giorgio Ghelli, and Andrew D. Gordon, **Secrecy and Group Creation**. Catuscia Palamidessi, editor. Proceedings of CONCUR 2000. Lecture Notes in Computer Science, Vol. 1877, Springer 2000, pp. 365-379.

[5] Luca Cardelli, Giorgio Ghelli, and Andrew D. Gordon, **Ambient Groups and Mobility Types.** Proceedings of IFIP TCS2000. J. van Leeuwen, O. Watanabe, M. Hagiya, P.D. Mosses, T. Ito (Eds.). Theoretical Computer Science; Exploring New Frontiers in Theoretical Informatics. Lecture Notes in Computer Science, Vol. 1872, Springer, 2000. pp. 333-347.

[6] Luca Cardelli and Andrew D. Gordon, **Mobile Ambients.** Foundations of Software Science and Computational Structures, Maurice Nivat (Ed.), Lecture Notes in Computer Science, Vol. 1378, Springer, 1998. pp. 140-155.

[7] Luca Cardelli and Andrew D. Gordon, **Anytime, Anywhere. Modal Logics for Mobile Ambients.** Proceedings of the 27th ACM Symposium on Principles of Programming Languages, 2000. pp 365-377.

[8] Silvano Dal Zilio, **Spatial Congruence for Ambients is Decidable**. Technical Report MSR-TR-2000-41, Microsoft Research, May 2000.

[9] Murdoch J. Gabbay, Andrew M. Pitts, **A New Approach to Abstract Syntax Involving Binders**. In Proceedings 14th Annual IEEE Symposium on Logic in Computer Science, Trento, Italy, July 1999. IEEE Computer Society Press, 1999. pp 214-224.

[10] Andrew D. Gordon and Luca Cardelli, **Equational Properties of Mobile Ambients.** Wolfgang Thomas, Editor. Foundations of Software Science and Computational Structures, Second International Conference, FOSSACS'99. Lecture Notes in Computer Science, Vol. 1578. Springer, 1999. pp 212-226.

[11] Robin Milner, **Communicating and Mobile Systems: the π-Calculus**. Cambridge University Press, 1999.

[12] Davide Sangiorgi, **Extensionality and Intensionality in the Ambient Logics**. Proc. POPL'01 (to appear).