A Predicate μ-Calculus for Mobile Ambients *

Huimin Lin
Laboratory for Computer Science
Institute of Software, Chinese Academy of Sciences
Email: lhm@ios.ac.cn

Abstract

Ambient logics have been proposed to describe properties for mobile agents which may evolve over time as well as space. This paper takes a predicate-based approach to extending an ambient logic with recursion, yielding a predicate μ-calculus in which fixpoint formulas are formed using predicate variables. An algorithm is developed for model checking finite-control mobile ambients against formulas of the logic, providing the first decidability result for model checking a spatial logic with recursion.

Keywords Model checking, Mobile ambients, Spatial logic, mu-Calculus, Fixpoints.

1 Introduction

The Ambient Calculus ([CG00a]) is a formal model for mobile computation in which processes reside and move in a hierarchy of named *locations* organised as a tree-like structure. The *ambient logic* is a first-order logic with modalities to express temporal as well as *spatial* evolution of mobile ambients [CG00b]. To have a taste of the logic, the formula $n[\mathsf{T}] \mid \Diamond\mathbf{0}$ is satisfied by a process consisting of two parallel subprocesses, one is at the location named n and the other may become void after some internal activities. To describe properties involving restricted names, the logic was extended in [CG01] to include a new kind of first-order quantification, the *fresh name quantification* $\text{И}x.A$ which is satisfied by a process P if for some name n not free in P and A, P satisfies $A[n/x]$ (the result of substituting each free occurrence of x in A with n). However, the logics proposed in [CG00b] are *finite* as they do not allow recursidive definitions.

The aim of this paper is two-fold: we will first add recursion to an ambient logic using a *predicate*-based approach, and then present a model-checking algorithm for the logic. The standard approach to extending modal logics with recursion is via *fixpoints*, as in the propositional μ-calculus. Taking this approach, fixpoint formulas are formed using recursion variables, together with recursor ν (and/or μ), as in $\nu X.A$ where X occurs free in A. In propositional μ-calculus the recursion body A is a proposition and X is a propositional variable. However, since ambient logics are *first-order*, not *propositional*, care must be exercised here. Being a first-order formula the recursion body A may contain free names. By fixpoint unfolding property, an occurrence of X in A stands for the entire formula, hence it should have the free names of A as free names. To facilitate the handling of free names in recursive formulas, we take a

*Supported by research grants from National Science Foundation of China and Chinese Academy of Sciences.

predicate-based approach, using *predicate variables*, instead of propositional variables, to form fixpoint formulas. Based on this idea we develop a predicate μ-calculus for the ambient calculus. It includes all the features of the ambient logics introduced in [CG00b, CG01]. A clear distinction is made between *propositions* and *predicates*. Propositions denote sets of processes, while predicates are interpreted as functions from names to sets of processes. Fixpoint formulas reside in the world of predicates. It is standard that such functions form a complete lattice with respect to the pointwise extension of the set inclusion ordering. We show that a positive predicate defines a monotonic functional over such a lattice, which guarantees the existences of both the greatest and the least fixpoints. The logic has all the desired properties listed in [CG00b, CG01], including the "Gabbay-Pitts property" for fresh name quantification.

With this logic we investigate how to model check finite-control ambients. Because of its practical importance, model checking has received active studies in the literature on ambient logics. The original paper [CG00b] contains an algorithm for model checking finite ambients against formulas without recursion nor fresh name quantification. The algorithm was extended in one direction to permit fresh name quantification in [CT01], and in another direction to allow recursively defined "finite-control" processes in [CGT02]. Techniques for handling spatial operators have been studied in detail and incorporated in the algorithms of [CG00b, CT01, CGT02]. Note that termination is not a problem for these algorithms because the logics in question are recursion-free. We shall show that model checking finite-control ambients against predicate μ-calculus (without composition adjunct) is decidable, by presenting a local-style algorithm. Since in our case both processes and formulas can be recursively defined, termination becomes a challenging issue. To establish a correct terminating condition we adapt the approach of [Win91], lifting it from the setting of propositional μ-calculus to predicate μ-calculus. The central idea is to generalise the syntax of a fixpoint formula to $\nu X\{I\}F$. The extra component $\{I\}$ serves as a history-recording device, used to remember the configurations that have been visited so far. When a configuration recorded in $\{I\}$ is encountered, the algorithm stops because a loop has been detected. Combined with the finite state property of finite-control ambients, such a strategy guarantees the termination of the algorithm. To the author's knowledge, this is the first decidability result for an ambient logic with recursion.

The rest of the paper is organised as follows: The ambient calculus is briefly reviewed in the following section; The language of predicate μ-calculus and its semantics is presented in Section 3; Section 4 is devoted to the model checking algorithm and its correctness proof; The paper is concluded with Section 5 where related work are also discussed.

For the sake of easy reading long proofs have been moved from the main body of the paper and collected in the appendix.

2 Ambient Calculus

Let $\mathcal{N}$ be a countable set of *names*, ranged over by $n,\ m,\ \ldots$, and V a set of *processes variables*, ranged over by $X,\ Y,\ Z,\ \ldots$. The language of ambient calculus is given in Figure 1.

The language has two binding constructs for names: both restriction $(\nu n)P$ and input $(n).P$ bind n in P. There is also a binder for processes variables: $\mathbf{fix}X.P$ binds X in P. These induce the usual notions of *bond* and *free* names/variables. The sets of bond and free names of P are denoted by $bn(P)$ and $fn(P)$, respectively, and the set of free process variables of P is $fpv(P)$. If $fpv(P)$ is empty then P is closed. The set of all closed processes is denoted $\mathcal{P}$. Processes

$M ::=$	capabilities							
$\mathsf{in}\ n$	can enter n	$\mid$	$\mathsf{out}\ n$	can exit n	$\mid$	$\mathsf{open}\ n$	can open n	$\mid$
ε	null path	$\mid$	$M.M$	composite path				

$P ::=$	processes							
$\mathbf{0}$	inactivity	$\mid$	$n[P]$	ambient	$\mid$	$M.P$	capability action	$\mid$
$P \mid Q$	composition	$\mid$	$(\nu n)P$	name restriction	$\mid$	$\langle M \rangle$	output	$\mid$
$(n).P$	input	$\mid$	X	variable	$\mid$	$\mathbf{fix}X.P$	recursion	

Figure 1: The Syntax of Ambient Calculus

which are different only in renaming of bound names/variables are called *α-equivalent*, and will be identified.

We use **fix** operator instead of replication because we shall be interested in "finite-control" processes [CGT02] when discussing model checking. It is easy to define replication in terms of **fix**: let P be a closed process, then $!P \stackrel{def}{=} \mathbf{fix}X.P \mid X$.

We write $P[M/n]$ to denote the result of capture-avoiding substitution of M for each free occurrence of n in P. This generalises to vectors in the usual way. If S is a set of processes and σ a substitution than $S\sigma = \{\, P\sigma \mid P \in S \,\}$. Substitutions which just interchange a pair of names are called *transpositions*, ranged over by τ. The transposition $[m, n/n, m]$ will be written as $[m \leftrightarrow n]$.

Structural congruence $\equiv$ is the least congruence relation generated from the rules in Figure 2. The reduction relation $\rightarrow$ is defined by the rules in Figure 3.

Some basic properties of reduction are listed in the following proposition:

Proposition 2.1 *For any processes P, Q and substitution σ,*

1. *$P \rightarrow Q$ implies $fn(Q) \subseteq fn(P)$.*
2. *$P \rightarrow Q$ implies $P\sigma \rightarrow Q\sigma$.*
3. *$P \equiv P'$ and $P \rightarrow Q$ implies $P' \rightarrow Q'$ for some $Q' \equiv Q$.*

When discussing model checking in Section 4 we shall be interested in processes from which only finite many states can be reached. The *finite-control* fragment of the ambient calculus is

StrRefl $P \equiv P$	StrSymm $P \equiv Q \Rightarrow Q \equiv P$
StrParZero $P \mid \mathbf{0} \equiv P$	StrTrans $P \equiv Q,\ Q \equiv R \Rightarrow P \equiv R$
StrParComm $P \mid Q \equiv Q \mid P$	StrParAss $P \mid (Q \mid R) \equiv (P \mid Q) \mid R$
StrResZero $(\nu n)\mathbf{0} \equiv \mathbf{0}$	StrResPar $n \notin fn(P) \Rightarrow (\nu n)(P \mid Q) \equiv P \mid (\nu n)Q$
StrFixSelf $\mathbf{fix}X.X \equiv \mathbf{0}$	StrResAmb $n \neq m \Rightarrow (\nu n)m[P] \equiv m[(\nu n)P]$
StrFixUnf $\mathbf{fix}X.P \equiv P[\mathbf{fix}X.P/X]$	StrResRes $(\nu n)(\nu m)P \equiv (\nu m)(\nu n)P$

Figure 2: Structural Congruence

$$\text{RedIn}\frac{}{n[\mathsf{in}\ m.P \mid Q] \mid m[R] \to m[n[P \mid Q] \mid R]} \qquad \text{RedRes}\frac{P \to Q}{(\nu n)P \to (\nu n)Q}$$

$$\text{RedOut}\frac{}{m[n[\mathsf{out}\ m.P \mid Q] \mid R] \to n[P \mid Q] \mid m[R]} \qquad \text{RedComm}\frac{}{(n).P \mid \langle M \rangle \to P[M/n]}$$

$$\text{RedOpen}\frac{}{\mathsf{open}\ n.P \mid n[Q] \to P \mid Q} \qquad \text{RedPar}\frac{P \to Q}{P \mid R \to Q \mid R}$$

$$\text{RedStr}\frac{P \equiv P',\ P' \to Q',\ Q' \equiv Q}{P \to Q} \qquad \text{RedAmb}\frac{P \to Q}{n[P] \to n[Q]}$$

Figure 3: Reduction Rules

defined in terms of a type system ([CGT02]). The type of a process is a non-negative integer which bounds the number of active ambients and outputs in the process. A *type environment* Γ is a finite set of pairs $\{(X_1, \tau_1),\ \ldots,\ (X_n, \tau_n)\}$ such that the X_i:s are distinct process variables and τ_i:s are non-negative integers. A *type judgment* is of the form $\Gamma \vdash P : \tau$ where P is a process and τ a natural number. The typing rules are listed in Figure 4. They are taken directly from [CGT02].

Definition 2.2 A process P is typable if for some Γ and τ, $\Gamma \vdash P :\ \tau$ can be derived from the typing rules in Figure 4.

A process P is finite-control if (i) only names can be output; (ii) for every recursive subterm $\mathbf{fix}X.Q$ in P, X is the only free variable of Q and it occurs only once in Q; (iii) P is typable. $\square$

The following result is quoted from [CGT02] (Theorem 1):

Theorem 2.3 *For any closed finite-control process P, there exist only finite many processes (up-to $\equiv$) reachable from P.*

$$\text{Var}\frac{(X,\tau) \in \Gamma}{\Gamma \vdash X :\ \tau} \qquad \text{Nil}\frac{}{\Gamma \vdash \mathbf{0} :\ 0}$$

$$\text{Par}\frac{\Gamma \vdash P :\ \tau,\ \Gamma \vdash Q :\ \tau'}{\Gamma \vdash P \mid Q :\ \tau + \tau'} \qquad \text{Res}\frac{\Gamma \vdash P :\ \tau}{\Gamma \vdash (\nu n)P :\ \tau}$$

$$\text{Output}\frac{}{\Gamma \vdash \langle n \rangle :\ 1} \qquad \text{Input}\frac{\Gamma \vdash P :\ \tau}{\Gamma \vdash (n).P :\ \max(\tau - 1, 1)}$$

$$\text{Move}\frac{\Gamma \vdash P :\ \tau,\ \mathsf{mv} \in \{\mathsf{in}\ ,\mathsf{out}\ \}}{\Gamma \vdash \mathsf{mv}\ n.P : \max(\tau, 1)} \qquad \text{Open}\frac{\Gamma \vdash P :\ \tau}{\Gamma \vdash \mathsf{open}\ n.P :\ \max(\tau - 1, 1)}$$

$$\text{Amb}\frac{\Gamma \vdash P :\ \tau}{\Gamma \vdash n[P] :\ \tau + 1} \qquad \text{Fix}\frac{\Gamma \cup \{(X,\tau)\} \vdash P :\ \tau',\ \tau' \leq \tau}{\Gamma \vdash \mathbf{fix}X.P :\ \tau}$$

Figure 4: Typing Rules

$A, B ::=$ propositions

$\top$	true	|	$\neg A$	negation	|
$A \wedge B$	conjunction	|	$\mathbf{0}$	void	|
$A \mid B$	composition	|	$A \triangleright B$	guarantee	|
$u[A]$	location	|	$A@u$	location adjunct	|
$u \circledR A$	revelation	|	$A \oslash u$	revelation adjunct	|
$\boxdot A$	next time modality	|	$\forall x.A$	universal quantification	|
$\text{И}x.A$	fresh name quantification	|	$F(\overline{u})$	application	

$F ::=$ predicates

X	predication variable	|	$(\overline{x})A$	abstraction	|	$\nu X.F$	greatest fixpoint

Figure 5: Formulas

3 The Logic

3.1 The Language

We assume a countably infinite set $\mathcal{V}$ of *name variables*, ranged over by $x,\ y,\ z, \ldots$, such that $\mathcal{V} \cap \mathcal{N} = \emptyset$, a countably infinite set $\mathcal{X}$ of *predicate variables*, ranged over also by $X,\ Y,\ Z,\ \ldots$[1]. Each predicate variable has associated with it an *arity* which is a natural number. We use $u,\ w,\ \ldots$ to range over $\mathcal{N} \cup \mathcal{V}$. The set of formulas of the logic is given by the BNF grammar in Figure 5

The syntax is divided into two categories: *propositions* and *predicates*. Semantically, propositions denote sets of processes, while predicates denote functions from sets of names to sets of processes. For propositions, apart from the usual propositional connectives $\top, \neg,\ \wedge$ and the universal quantification, we also have operators which talk about the structures of processes: $\mathbf{0}$ is satisfied by any process which is structurally congruent to $\mathbf{0}$; $A \mid B$ is satisfied by any process which can be decomposed into two parallel processes, one satisfying A and the other satisfying B; $\triangleright$ is composition adjunct: P satisfies $A \triangleright B$ if $P \mid Q$ satisfies B for any Q satisfying A; $n[A]$ indicates that the computation is located at n: P satisfies $n[A]$ if $P \equiv n[Q]$ and Q satisfies A; the adjunct of the location modality is $A@n$ which is satisfied by any process P such that $n[P]$ satisfies A; $n \circledR A$ is used to reveal a private name: P satisfies $n@A$ if $P \equiv (\nu n)Q$ and Q has property A; the adjunct to revelation is $A \oslash n$ which is satisfied by any process P such that $(\nu n)P$ satisfies A; $\text{И}x.A$ is fresh name quantification: it is satisfied by such processes P which satisfies $A[n/x]$ for some name n fresh with respect to both A and P. There is also a next time modality $\boxdot A$: P satisfies $\boxdot A$ if whenever $P \to Q$ then Q has property A.

A predicate is either a predicate variable X, or an abstraction $(\overline{x})A$, or a greatest fixpoint $\nu X.F$. When forming an abstraction $(\overline{x})A$ it is required that $\overline{x}$ be a vector of distinct name variables. The arity of a predicate F is defined thus: the arity of X if F has the form X or $\nu X.F$, or the length of $\overline{x}$ if F has the form $(\overline{x})A$. In abstractions and applications we always require arities to be matched properly. By "formulas" we refer to either propositions or predicates. We will use W to range over formulas.

[1]Process variables and predicate variables can always be differentiated by context.

The *sometime* and *somewhere* modalities, present in the ambient logics of [CG00b, CT01, CGT02], are not taken as primitive here because they can be defined from fixpoints and other operators, as we shall see later.

In $\forall x A$, $\text{И} x A$ and $(\overline{x})A$, x and $\overline{x}$ are binding with scope A. In $\nu X.F$ the predicate variable X is binding with scope F. These introduce the notions of *bound* and *free name variables* as well as *bound* and *free predicate variables* in the usual way. The set of free names, free name variables and free predicate variables of a formula A are denoted by $fn(A)$, $fnv(A)$ and $fpv(A)$, respectively. Formulas that do not have free name variables are *name-closed.* Note that name-closed formulas may still have free names, and we call formulas which have neither free name variables nor free names *completely name-closed.* Formulas that do not have free predicate variables are *predicate-closed.* A formula is *closed* if it is name-closed as well as predicate-closed.

Formulas which differ only in renaming of bound variables are α-equivalent, denoted by $\equiv_\alpha$. We extend the notion of name substitutions to allow not only names but also name variables to be replaced by names. We shall identify $((\overline{x})A)(\overline{u})$ with $A[\overline{u}/\overline{x}]$, the outcome of capture-avoiding substitution of $\overline{x}$ with $\overline{u}$ in A.

The unary operator $\neg$ is *negative.* For any B, $\_ \triangleright B$ can be regarded as a unary operator which is also negative. An occurrence of a predicate variable is positive if it is under an even number of negative operators. X occurs *positively* in a formula W if every occurrence of X in W is positive. Otherwise we say X occurs *negatively* in W.

A fixpoint predicate $\nu X.F$ is *well-formed* if F is completely name-closed and X occurs positively in F. A formula is well-formed if every fixpoint subformula in it is well-formed. We shall only consider well-formed formulas in this paper.

The requirement for predicates to be completely name-closed forces fixpoint formulas to be *fully parameterised*, in the sense that the sets of free names and free name variables of any recursively defined proposition $F(\overline{u})$ are determined solely by the actual parameter part $\overline{u}$, without affected by the body F. Thus there are no"global references" in recursive formulas, using a jargon from the programming language world. This eliminates any confusion in determining free names for a formula, and allows a clean semantics to be developed for the logic.

3.2 Semantics

Semantics will be given to name-closed formulas which may have free predicate variables. To interpret such formulas we need *predicate valuations.* A predicate valuation v assigns to every predicate variable X of arity k a function $v(X) : \mathcal{N}^k \to 2^{\mathcal{P}}$. For each k, let $f \sqsubseteq^k g$ if and only if $f(\overline{n}) \subseteq g(\overline{n})$ for any $\overline{n} \in \mathcal{N}^k$. Then $(\{f : \mathcal{N}^k \to 2^{\mathcal{P}}\}, \sqsubseteq^k)$ is a complete lattice, with meet $\sqcap^k$ and join $\sqcup^k$. We shall omit superscript k and simply write $\sqsubseteq$, $\sqcap$ and $\sqcup$ when no confusion may arise.

The denotation of formulas is defined inductively in Figure 6, where a proposition is interpreted as an element of $2^{\mathcal{P}}$ and a predicate of arity k is interpreted as an element in the function space $\mathcal{N}^k \to 2^{\mathcal{P}}$.

If A is predicate-closed then $[\![A]\!]_v$ does not depend on v and will be written $[\![A]\!]$. We will write $P \models_v A$ to mean $P \in [\![A]\!]_v$.

The following proposition justifies our decision to identify α-equivalent formulas.

Proposition 3.1 *If $W \equiv_\alpha W'$ then $[\![W]\!]_v = [\![W']\!]_v$ for any v.*

An important property of the semantics is monotonicity:

$$
\begin{array}{lcl}
[\![\mathsf{T}]\!]_v & = & \mathcal{P} \\
[\![\neg A]\!]_v & = & \mathcal{P} - [\![A]\!]_v \\
[\![A \wedge B]\!]_v & = & [\![A]\!]_v \cap [\![B]\!]_v \\
[\![\mathbf{0}]\!]_v & = & \{\, P \mid P \equiv \mathbf{0} \,\} \\
[\![A \mid B]\!]_v & = & \{\, P \mid P \equiv Q \mid R \ \text{ and } \ Q \in [\![A]\!]_v,\ R \in [\![B]\!]_v \,\} \\
[\![A \triangleright B]\!]_v & = & \{\, P \mid \textit{if } Q \in [\![A]\!]_v \textit{ then } Q \mid P \in [\![B]\!]_v \,\} \\
[\![n[A]]\!]_v & = & \{\, P \mid P \equiv n[Q] \text{ for some } Q \in [\![A]\!]_v \,\} \\
[\![A@n]\!]_v & = & \{\, P \mid n[P] \in [\![A]\!]_v \,\} \\
[\![n \circledR A]\!]_v & = & \{\, P \mid P \equiv (\nu n)Q \text{ for some } Q \in [\![A]\!]_v \,\} \\
[\![A \oslash n]\!]_v & = & \{\, P \mid (\nu n)P \in [\![A]\!]_v \,\} \\
[\![\boxdot A]\!]_v & = & \{\, P \mid \text{for any } Q,\ P \to Q \text{ implies } Q \in [\![A]\!]_v \,\} \\
[\![\text{И}x.A]\!]_v & = & \bigcup_{n \notin \mathit{fn}(A)} \{\, P \mid P \in [\![A[n/x]]\!]_v \ \text{ and } \ n \notin \mathit{fn}(P) \,\} \\
[\![\forall x.A]\!]_v & = & \bigcap_n [\![A[n/x]]\!]_v \\
[\![F(\overline{n})]\!]_v & = & [\![F]\!]_v((\overline{n})) \\
[\![X]\!]_v & = & v(X) \\
[\![(\overline{x})A]\!]_v & = & \lambda\overline{n}.[\![A[\overline{n}/\overline{x}]]\!]_v \\
[\![\nu X.F]\!]_v & = & \bigsqcup \{\, f \mid f \sqsubseteq [\![F]\!]_{v[X \mapsto f]} \,\}
\end{array}
$$

Figure 6: Interpretation of Formulas

Theorem 3.2 *1. Suppose X is positive in A and F. Then $f \sqsubseteq g$ implies*

(a) $[\![A]\!]_{v[X\mapsto f]} \subseteq [\![A]\!]_{v[X\mapsto g]}$

(b) $[\![F]\!]_{v[X\mapsto f]} \sqsubseteq [\![F]\!]_{v[X\mapsto g]}$

2. Suppose X is negative in A and F. Then $f \sqsubseteq g$ implies

(a) $[\![A]\!]_{v[X\mapsto g]} \subseteq [\![A]\!]_{v[X\mapsto f]}$

(b) $[\![F]\!]_{v[X\mapsto g]} \sqsubseteq [\![F]\!]_{v[X\mapsto f]}$

Corollary 3.3 *$[\![\nu X.F]\!]_v$ is the greatest fixpoint of the functional $\lambda f.[\![F]\!]_{v[X\mapsto f]}$.*

Now we set to establish some important results concerning *fresh names*. A name is fresh with respect to a process P and a proposition A if it does not occur free in P and A. It turns out that transpositions are a useful instrument in proving properties concerning fresh names.

Definition 3.4 A transposition τ is preserved by a function $f : \overline{\mathcal{N}} \to 2^{\mathcal{P}}$ if $(f(\overline{n}))\tau = f(\overline{n}\tau)$ for any $\overline{n}$. A transposition τ is preserved a valuation v on a set of predicate variables $\mathcal{U}$ if τ is preserved by $v(X)$ for every $X \in \mathcal{U}$. □

Proposition 3.5 *1. If τ is preserved by v on $\mathit{fpv}(A)$ then $([\![A]\!]_v)\tau = [\![A\tau]\!]_v$.*

2. If τ is preserved by v on $\mathit{fpv}(F)$ then τ is preserved by $[\![F]\!]_v$.

The semantic definition of the fresh renaming quantification is stated in "existential" style: P satisfies $\text{И}x.A$ if there is some fresh name n such that P satisfies $A[n/x]$. Since n is not free in either P or A, this particular choice of n should not matter: any other name m with $m \notin fn(P, A)$ should equally do. Thus the semantics of the fresh renaming quantifier can also be characterised "universally":

Proposition 3.6 *$P \in [\![A[m/x]]\!]_v$ for some m iff $P \in [\![A[n/x]]\!]_v$ for every $n \notin fn(P, A)$ such that $[m\leftrightarrow n]$ is preserved by v on $fpv(A)$.*

Proof: The "if" direction is trivial. For the "only if" direction, suppose $P \in [\![A[m/x]]\!]_v$ for some $m \notin fn(P, A)$. For any n such that $n \notin fn(P, A)$ and v is $[m\leftrightarrow n]$-preserving on $fpv(A)$, we have

$$\begin{array}{rcll} [\![A[n/x]]\!]_v & = & [\![A[m/x][m\leftrightarrow n]]\!]_v & (m, n \notin fn(A)) \\ & = & ([\![A[m/x]]\!]_v)[m\leftrightarrow n] & (\text{Proposition 3.5}) \end{array}$$

Since $P \in [\![A[m/x]]\!]_v$, $P[m\leftrightarrow n] \in ([\![A[m/x]]\!])[m\leftrightarrow n]$. Since $m, n \notin fn(P)$, $P[m\leftrightarrow n] \equiv P$. Therefore $P \in [\![A[n/x]]\!]_v$. $\Box$

Corollary 3.7 *(Gabbay-Pitts property) For closed formula $\text{И}x.A$, $P \in [\![\text{И}x.A]\!]$ iff $P \in [\![A[n/x]]\!]$ for every $n \notin fn(P, A)$.*

According to the semantics of $\forall xA$, to check if $P \in [\![\forall xA]\!]_v$ requires to instantiate x with every name. However, as the following proposition demonstrates, it is sufficient to consider only the free names of A, plus one fresh name. This finite characterisation will be exploited in the model checking algorithm in Section 4.

Proposition 3.8 *For closed formula $\forall x.A$, $P \in [\![\forall x.A]\!]$ iff there exists $m \notin fn(P, A)$ such that $P \in \bigcap_{k \in fn(P,A)\cup\{m\}} [\![A[k/x]]\!]$.*

Proof: The "only if" direction is trivial. For the "if" direction, assume $P \in \bigcap_{k\in fn(P,A)\cup\{m\}} [\![A[k/x]]\!]$ for some $m \notin fn(P, A)$, we show $P \in [\![A[n/x]]\!]$ for any n. This is immediate if $n \in fn(P, A)\cup\{m\}$. Now suppose $n \notin fn(P, A)\cup\{m\}$. Then $[\![A[n/x]]\!] = [\![A[m/x][m\leftrightarrow n]]\!] = ([\![A[m/x]]\!])[m\leftrightarrow n]$ by Proposition 3.5. Since $P \in [\![A[m/x]]\!]$, $P[m\leftrightarrow n] \in ([\![A[m/x]]\!])[m\leftrightarrow n]$. Since $m, n \notin fn(P)$, $P[m\leftrightarrow n] \equiv P$. Therefore $P \in [\![A[n/x]]\!]$. $\Box$

3.3 Definable Operators

If $fnv(A) = \{\overline{x}\}$, the actual parameters to every free occurrences of X in A is $\overline{x}$, and X does not occur within the scope of any name quantifier, then we shall abbreviate $(\nu X.(\overline{x})A)(\overline{n})$ to $\nu X.A'[\overline{n}/\overline{x}]$ where A' is obtained from A by replacing $X(\overline{x})$ with X. For example, $(\nu X.(x, y)(x[\mathsf{T}] \mid y®X(x, y)))(m, n)$ will be written as $\nu X.m[\mathsf{T}] \mid n®X$.

The "anytime" modality $\Box$ of [CG00b] can be defined from $\boxdot$ and the greatest fixpoint:

$$\Box A \stackrel{def}{=} \nu X.A \wedge \boxdot X$$

The duals of $\boxdot$ and $\Box$ are $\diamonddot$ and $\Diamond$, defined respectively:

$$\diamonddot A \stackrel{def}{=} \neg\boxdot\neg A \qquad \Diamond A \stackrel{def}{=} \neg\Box\neg A$$

The least fixpoint is the dual of the greatest fixpoint:

$$\mu X.F \stackrel{def}{=} \neg\nu X.\neg F[\neg X/X]$$

where negation on abstractions is defined by letting $\neg(\overline{x})A = (\overline{x})(\neg A)$. It is easy to check that $\Diamond A = \mu X.A \vee \Diamond X$.

The "somewhere one step downward in space" modality and its dual "anywhere one step downward in space" modality are:

$$\diamond A \stackrel{def}{=} \exists x.x[A] \mid \mathsf{T} \qquad \boxtimes A \stackrel{def}{=} \neg\diamond\neg A$$

Now the "somewhere" and "anywhere" modalities $\diamond$ and $\boxtimes$ of [CG00b] can be defined:

$$\diamond A \stackrel{def}{=} \mu X.A \vee \diamond X \qquad \boxtimes A \stackrel{def}{=} \neg\diamond\neg A$$

where X does not occur free in A.

Combining fresh name quantification and revelation gives rise to "hidden name quantification":

$$\mathsf{H}x.A \stackrel{def}{=} \mathsf{N}x.x\circledR A$$

And we have, as expected,

$$[\![\mathsf{H}x.A]\!]_v = \{\, P \mid P \equiv (\nu n)Q \text{ for some } n \notin fn(A) \text{ and } Q \in [\![A[n/x]]\!]_v \,\}$$

4 Model Checking

This section is devoted to developing a model checker for our logic. Since model checking for finite ambient logic with composition adjunct is already undecidable [CT01], we shall confine to the $\triangleright$-free subset of the logic. On the process model side, we shall restrict to the "finite-control" ambients of [CGT02]. Model checking algorithms can be divided into two categories: *global* and *local*. The global algorithms require *a prior* construction of state space, which is infeasible in our setting because, for instance, we can not determine which "fresh name" should be chosen to instantiate the x in $\mathsf{N}xA$ before model checking starts. Our algorithm will be a "lifting" of the local model checking algorithm for the propositional μ-calculus proposed in [Win91] to the predicate μ-calculus, with features to deal with spatial modalities.

Given a finite set of names $\mathcal{N}_0 \subset \mathcal{N}$ and a finite set of closed processes $\mathcal{P}_0 \subseteq \mathcal{P}$, let $I = \{(\overline{n}_1, \overline{P}_1),\ \ldots,\ (\overline{n}_l, \overline{P}_l)\}$ where $\overline{n}_i$:s are vectors of the same length over $\mathcal{N}_0$, $\overline{n}_i \neq \overline{n}_j$ for $i \neq j$, and $\overline{P}_i \subseteq \mathcal{P}_0$. I can be regarded as a function from $\mathcal{N}^k$ to $2^{\mathcal{P}}$, where k (the "arity" of I) is the length of $\overline{n}_1$, by letting

$$I(\overline{n}) = \begin{cases} \{\overline{P}\} & \text{if for some } \overline{P},\ (\overline{n}, \overline{P}) \in I \\ \emptyset & \text{if for no } \overline{P},\ (\overline{n}, \overline{P}) \in I \end{cases}$$

We generalise the syntax of fixpoint predicates to $\nu X\{I\}F$, where the arity of I equals to that of X, and regard νXF as an abbreviation of $\nu X\{\emptyset\}F$. The $\{I\}$ component is a history-recording device to facilitate loop detection during model checking, and will play a key role in establishing the termination condition for the algorithm. It does not contribute to the set of free names of a formula.

The interpretation of the generalised fixpoint predicates is

$$[\![\nu X.\{I\}F]\!]_v = \sqcup\{\, f \mid f \sqsubseteq I \sqcup [\![F]\!]_{v[X\mapsto f]}\,\}$$

Since X occurs positively in F, $\lambda g.I \sqcup [\![F]\!]_{v[X\mapsto g]}$ is a monotonic functional over $\mathcal{N}^k \to 2^{\mathcal{P}}$, and $[\![\nu X.\{I\}F]\!]_v$ is its greatest fixpoint, denoted $\nu g.I \sqcup [\![F]\!]_{v[X\mapsto g]}$.

Lemma 4.1 *Let $\zeta(g)$ be a monotonic function on $\mathcal{N}^k \to 2^{\mathcal{P}}$. For any $f : \mathcal{N}^k \to 2^{\mathcal{P}}$, $f \sqsubseteq \nu g.\zeta(g)$ iff $f \sqsubseteq \zeta(\nu g.f \sqcup \zeta(g))$.*

Proof: ($\Rightarrow$) Assume $f \sqsubseteq \nu g.\zeta(g)$. Then

$$\nu g.\zeta(g) = f \sqcup \nu g.\zeta(g) = f \sqcup \zeta(\nu g.\zeta(g))$$

That is, $\nu g.\zeta(g)$ is a post-fixpoint of the monotonic function $\lambda g.f \sqcup \zeta(g)$. Hence $\nu g.\zeta(g) \sqsubseteq \nu g.f \sqcup \zeta(g)$. Therefore

$$\begin{array}{rll} f \sqsubseteq \nu g.\zeta(g) = & \zeta(\nu g.\zeta(g)) & \text{(fixpoint unfolding)} \\ \sqsubseteq & \zeta(\nu g.f \sqcup \zeta(g)) & (\zeta \text{ monotonic}) \end{array}$$

($\Leftarrow$) Assume $f \sqsubseteq \zeta(\nu g.f \sqcup \zeta(g))$. We have

$$\nu g.f \sqcup \zeta(g) = f \sqcup \zeta(\nu g.f \sqcup \zeta(g)) = \zeta(\nu g.f \sqcup \zeta(g))$$

Thus $\nu g.f \sqcup \zeta(g)$ is a post-fixpoint of ζ. So $\nu g.f \sqcup \zeta(g) \sqsubseteq \nu g.\zeta(g)$. Therefore

$$f \sqsubseteq f \sqcup \zeta(\nu g.f \sqcup \zeta(g)) = \nu g.f \sqcup \zeta(g) \sqsubseteq \nu g.\zeta(g)$$

□

Lemma 4.2 *If $P \in I(\overline{n})$ then $P \in [\![\nu X.\{I\}F]\!]_v(\overline{n})$; Otherwise $P \in [\![\nu X.\{I\}F]\!]_v(\overline{n})$ iff $P \in [\![F[\nu X.\{(\overline{n},\{P\}) \sqcup I\}F/X]]\!]_v(\overline{n})$.*

Proof: Since $I \sqsubseteq I \sqcup [\![F]\!]_{v[X\mapsto f]}$, $I \sqsubseteq [\![\nu X.\{I\}F]\!]_v$ by definition. Therefore if $P \in I(\overline{n})$ then $P \in [\![\nu X.\{I\}F]\!]_v(\overline{n}) = [\![\nu X.\{I\}F(\overline{n})]\!]_v$.

If $P \notin I(\overline{n})$ then

$$\begin{array}{lll} & P \in [\![\nu X.\{I\}F]\!]_v(\overline{n}) & \\ \iff & P \in (\nu g.I \sqcup [\![F]\!]_{v[X\mapsto g]})(\overline{n}) & \\ \iff & (\overline{n},\{P\}) \sqsubseteq \nu g.I \sqcup [\![F]\!]_{v[X\mapsto g]} & \\ \iff & (\overline{n},\{P\}) \sqsubseteq I \sqcup [\![F]\!]_{v[X\mapsto \nu g.\{(\overline{n},\{P\})\}\sqcup I \sqcup [\![F]\!]_{v[X\mapsto g]}]} & \text{(Lemma 4.1)} \\ \iff & (\overline{n},\{P\}) \sqsubseteq [\![F]\!]_{v[X\mapsto \nu g.\{(\overline{n},\{P\})\}\sqcup I \sqcup [\![F]\!]_{v[X\mapsto g]}]} & (P \notin I(\overline{n})) \\ \iff & (\overline{n},\{P\}) \sqsubseteq [\![F]\!]_{v[X\mapsto [\![\nu X\{(\overline{n},\{P\})\sqcup I\}F]\!]_v]} & \text{(definition of } [\![\nu X\{I\}F]\!]_v) \\ \iff & (\overline{n},\{P\}) \sqsubseteq [\![F[\nu X\{(\overline{n},\{P\}) \sqcup I\}F/X]]\!]_v & \\ \iff & P \in [\![F[\nu X\{(\overline{n},\{P\}) \sqcup I\}F/X]]\!]_v(\overline{n}) & \end{array}$$

□

When model checking a recursively defined property, a crucial point is to decide when to stop. Lemma 4.2 provides us with a termination strategy: when encountering a fixpoint formula $(\nu X.\{I\}F)(\overline{n})$ and the present process is P, we first check if $P \in I(\overline{n})$, *i.e.* if the configuration $(P, (\nu X.\{I\}F)(\overline{n}))$ has been met before. If it is the case then a loop is detected and we can stop with *true* (due to the nature of greatest fixpoint); Otherwise we remember this configuration by recording $(\overline{n}, \{P\})$ in the history associated to the fixpoint formula, and continue to check the property by unfolding the fixpoint. Since the process in question is finite state, the formula can only be unfolded finite many times before a loop is detected.

For technical convenience, in the rest of this section we shall assume all bound names in a process have been renamed apart. When dealing with revelation we need to pull restrictions to the top level of a process. This is accomplished by the relation $\succ_n$ defined by the following rules:

$$\text{Rev}\frac{}{(\nu n)P \succ_n P} \qquad \text{RevPar}\frac{P \succ_n P', \quad n \notin fn(Q)}{P \mid Q \succ_n P' \mid Q}$$

$$\text{RevRes}\frac{P \succ_n P', \quad m \neq n}{(\nu m)P \succ_n (\nu m)P'} \qquad \text{RevAmb}\frac{P \succ_n P', \quad m \neq n}{m[P] \succ_n m[P']}$$

$$\text{RevFix}\frac{P[\mathbf{fix}X.P/X] \succ_n P'}{\mathbf{fix}.P \succ_n P'}$$

Lemma 4.3 *Suppose all bound names in P are different. Then $P \equiv (\nu n)Q$ iff either $P \succ_n Q$ or $P \equiv Q$ and $n \notin fn(P)$.*

Proof: (Sketch) ($\Longleftarrow$) If $P \equiv Q$ and $n \notin fn(P)$ then $P \equiv P \mid \mathbf{0} \equiv P \mid (\nu n)\mathbf{0} \equiv (\nu n)(P \mid \mathbf{0}) \equiv (\nu n)P \equiv (\nu n)Q$. Otherwise assume $P \succ_n Q$, and we can show $P \equiv (\nu n)Q$ by induction on the inference of $P \succ_n Q$.
($\Longrightarrow$) By induction on why $P \equiv (\nu n)Q$. There are five clauses in the definition of structural congruence (Figure 2) for deriving this, each corresponding to a rule defining $P \succ_n Q$. $\Box$
It is known from the results of [CG00b, CT01, CGT02] that, for any process P, $P \equiv \mathbf{0}$ can be tested in polynomial time, the set $\{ Q \mid P \rightarrow Q \}$ can be computed in polynomial time, the set $\{ (P_1, P_2) \mid P \equiv P_1 \mid P_2 \}$ can be computed in polynomial space, and given name n, $P \equiv n[Q]$ for some Q can be tested in polynomial time. Now we show that all the pairs (n, Q) such that $P \succ_n Q$ can also be computed in polynomial time.

Proposition 4.4 *For any P, we can compute in polynomial time all the pairs (n, Q) such that $P \succ_n Q$.*

Proof: (Sketch) We first replace by $\mathbf{0}$ all subterms of P which are structurally congruent to $\mathbf{0}$. Let the resulting term be P'. Then we rename apart all bound names in P'. Next we start from the leafs in the syntax tree for P', moving upwards and collecting terms of the form $(\nu n)Q$ according to the rules defining $\succ_n$; when encountering operators without defining rules we simply discard what have been collected. Clearly this last step can be done in time linear to the length of P' (hence to the length of P). $\Box$

Next we give a bound on the number of names needed for model checking if process P has property A. We may assume all bound names in P and all bound name variables in A are

$\mathtt{check}(P, A) = \text{case } A \text{ of}$
$\top \Rightarrow \mathit{true}$
$\mid \neg B \Rightarrow \neg\mathtt{check}(P, B)$
$\mid B \wedge C \Rightarrow \mathtt{check}(P, B) \wedge \mathtt{check}(P, C)$
$\mid \mathbf{0} \Rightarrow \text{if } P \equiv \mathbf{0} \text{ then } \mathit{true} \text{ else } \mathit{false}$
$\mid B \mid C \Rightarrow \bigvee_{P_1 \mid P_2 \equiv P} \mathtt{check}(P_1, B) \wedge \mathtt{check}(P_2, C)$
$\mid n[B] \Rightarrow \text{if } P \equiv n[Q] \text{ then } \mathtt{check}(Q, B) \text{ else } \mathit{false}$
$\mid B@n \Rightarrow \mathtt{check}(n[P], B)$
$\mid n®B \Rightarrow \text{ if } n \in \mathit{fn}(P) \text{ then } \mathit{false} \text{ else } \bigvee_{P \succ_m Q} \mathtt{check}(Q[n/m], B) \vee \mathtt{check}(P, B)$
$\mid B \oslash n \Rightarrow \mathtt{check}((\nu n)P, B)$
$\mid \boxdot B \Rightarrow \bigwedge_{P \to Q} \mathtt{check}(Q, B)$
$\mid \forall x B \Rightarrow \bigwedge_{n \in \mathit{fn}(P,A) \cup \{\mathtt{new}(\mathit{fn}(P,A))\}} \mathtt{check}(P, B[n/x])$
$\mid \text{И}x B \Rightarrow \mathtt{check}(P, B[\mathtt{new}(\mathit{fn}(P, A))/x])$
$\mid (\nu X\{I\}F)(\overline{n}) \Rightarrow \text{if } P \in I(\overline{n}) \text{ then } \mathit{true} \text{ else } \mathtt{check}(P, F[\nu X\{\{(\overline{n}, \{P\})\} \bigsqcup I\}F/X](\overline{n}))$

Figure 7: The Algorithm

different. Let $\natural_P$ be the number of names appearing in P and $\natural_A$ the number of names and name variables appearing in A (the $\{I\}$ component of any generalised fixpoint subformula does not count). Let $\natural_{P,A} = \natural_P + \natural_A$ and $N_{P,A} = \{n_1, n_2, \ldots, n_{\natural_{P,A}+1}\}$, ordered as such. Without loss of generality, we may assume $\mathit{fn}(P, A) \subset N_{P,A}$. For a set of names $V \subset N_{P,A}$, function $\mathtt{new}(V)$ returns the least name in $N_{P,A} - V$. The algorithm is presented in Figure 7.

Lemma 4.5 *For every recursive call of* `check` *with actual parameters* (P', A') *in the body of* $\mathtt{check}(P, A)$*,* $\natural_{P',A'} \leq \natural_{P,A}$.

Proof: (Sketch) Each recursive call is on a subprocess of P and/or a subformula of A, except for the cases of $\boxdot$ and ν. In the case of $\boxdot$, by Proposition 2.1 we have $\mathit{fn}(Q) \subseteq \mathit{fn}(P)$ for any Q such that $P \to Q$; In the case of ν we have $\mathit{fn}((\nu X\{I\}F)(\overline{n})) = \{\overline{n}\} = \mathit{fn}(F[\nu X\{\{(\overline{n}, \{P\})\} \bigsqcup I\}F/X](\overline{n}))$ $\square$

Theorem 4.6 *For any finite-control process P and closed formula A not containing composition adjunct,* $\mathtt{check}(P, A)$ *terminates, and it returns true iff* $P \models A$.

Proof: (Sketch) By Lemma 4.5 the names used by the algorithm is confined to $N_{P,A}$. Define the relation $\prec$ on formulas thus: $C \prec B$ if either C is a proper subformula of B, or $C \equiv (\nu X\{\{(\overline{n}, \{Q\})\} \bigsqcup I\}F)(\overline{n})$ and $B \equiv (\nu X\{I\}F)(\overline{n})$ with $Q \notin I(\overline{n})$. The relation is well-founded because the number of processes reachable from P is finite. We apply well-founded induction on $\prec$, examining the cases in the body of $\mathtt{check}(P, A)$. Each case leads to a call or calls of `check` on formulas smaller than A with respect to $\prec$, thus termination is guaranteed. For correctness, all the cases except the last follow closely the interpretation of formulas in Figure 6 (the case for $n®B$ is justified by Lemma 4.3, and the cases of $\forall x.B$ and $\text{И}x.B$ by Proposition 3.8 and Corollary 3.7, respectively). For the last case when A is $(\nu X\{I\}F)(\overline{n})$, Lemma 4.2 applies. $\square$

Corollary 4.7 *Model checking for finite-control processes against predicate spatial μ-calculus without composition adjunct is decidable.*

5 Conclusion

We have presented a first-order μ-calculus for mobile ambients. It extends the standard finite ambient logic with fixpoints. In our framework fixpoint formulas are formed in terms of predicates which are functions from names to propositions. A model checking algorithm has been developed for the logic (without composition adjunct) over finite-control ambients. It is the first model checker for a spatial logic and an ambient calculus both of which admit recursion. The algorithm has been impletemented and tested on some typical examples.

The ambient logic was originally proposed in [CG00b]. Fresh name quantification was introduced to the logic in [CG01], as a means to talk about restricted names. In the other direction, [CGh01] and [Zil01] add fixpoints to subsets of the logic without the modalities for private names. A spatial logic for asynchronous π-calculus was introduced in [CC01], which also has fresh name quantification and recursion. Modulo the differences on the underlining process models, the main difference between that logic and the logic presented in this paper lies in the formation of fixpoint formulas which has consequences to the development of semantics. Fixpoints are formed using propositional variables in [CC01]. Since propositional variables do not syntactically have name parameters, while the interpretation of fresh name quantification demands them, as a remedy, the notion of *Psets* ("property sets") are introduced. A Pset is a set of processes equipped with a "support" which is a finite set of names. A valuation assigns to each propositional variable a Pset the support of which plays the role of "free names" for the variable. The set of free names of a formula then depends on a valuation (which is a semantic device), hence can not be determined syntactically. Our framework is predicate-based. Predicate variables take parameters (names), so free names of formulas are manifest in syntax. This avoids the need for property sets and leads to a simpler semantics.

Model checking algorithms for finite ambients and finite ambient logics have been presented in [CG00b, CT01]. The complexities were analysed in [CZGMT01]. The notion of "finite-control" ambients, i.e. finite-state ambients, was introduced in [CGT02] where the algorithm of [CT01] was extended to cover this class of recursively defined processes (the logic remains finite). The issue of model checking was not addressed in [CC01]. In the π-calculus world, model checking finite-control processes against a temporal (but not spatial) logic was investigated in [Dam96].

There are several directions for further research. The model checking algorithm of Section 4 has been designed mainly with clarity and conciseness, but not efficiency, in mind, and we are currently working on more efficient algorithms for the logic. Validity and inference systems for the logic, particularly inference rules for recursion, are worth investigating [CG00b, CG01, CC02]. Characterisations of behavioural equivalences using a finite ambient logic have been studied in [San01, HLS02], and it will be interesting to see to what extent these results can be strengthened by having recursion in the logic.

References

[CC01] Luis Caires and Luca Cardeli. A Spatial Logic for Concurrency (Part I). TACS'2001.

[CC02] Luis Caires and Luca Cardeli. A Spatial Logic for Concurrency (Part II). CONCUR'2002.

[CGh01] L. Cardelli and G. Ghelli. A Query Language Based on the Ambient Logic. ESOP'2001, LNCS 2028, pp.1-22. Springer-Verlag, 2001.

[CG00b] L. Cardelli and A. Gordon. Anytime, Anywhere: Modal logics for mobile ambients. POPL'2000, pp.365-377. ACM Press, Jan. 2000.

[CG00a] L. Cardelli and A. Gordon. Mobile ambients. *Theoretical Computer Science*, 240:177-213. Elsevier, 2000.

[CG01] L. Cardelli and A. Gordon. Logical Properties of Name Restriction. Typed Lambda Calculi and Applications, LNCS 2044. Springer-Verlag, 2001.

[CT01] W. Charatonik and J.-M. Talbot. The Decidability of Model Checking Mobile Ambients. 15th Annual Conf. of European Association for Computer Science Logic. LNCS 2142, pp.339-354. Springer-Verlag, 2001.

[CGT02] W. Charatonik, A. Gordon and J.-M. Talbot. Finite-Control Mobile Ambients. ESOP'02, LNCS . Springer-Verlag, 2002.

[CZGMT01] W. Charatonik, S. Dal Zilio, A. Gordon, S. Mukhopadhyay and J.-M. Talbot. The Complexity of Model Checking Ambients. In FoSSaCS 2001, F. Honsell, M. Miculan (Eds.), LNCS 2030, pp. 152-167. Springer-Verlag, 2001.

[Dam96] M. Dam. Model Checking Mobile Processes. *Information and Computation* 129:25–51. 1996.

[HLS02] D. Hirschkoff, E. Lozes and D. Sangiorgi. Separability, expressiveness, and decidability in the Ambient Logic. LICS'2002.

[San01] D. Sangiorgi. Extensionality and Intensionality of the Ambient Logics. POPL'2001, pp.4-13. ACM Press, 2001.

[Win91] G. Winskel. A Note on Model Checking the Modal μ-Calculus. *Theoretical Computer Science* 83:157-167. Elsevier, 1991.

[Zil01] S. Dal Zilio. Fixed Points in Ambient Logic. 2001.

A Proofs

Proposition 3.1 If $W \equiv_\alpha W'$ then $[\![W]\!]_v = [\![W']\!]_v$ for any v.
Proof: By induction on the structure of W.

- $W \equiv \mathsf{T}$. Trivial.
- $W \equiv B \wedge C$. Then $W' \equiv B' \wedge C'$ for some $B' \equiv_\alpha B$ and $C' \equiv_\alpha C$. By the induction hypothesis, $[\![B]\!]_v = [\![B']\!]_v$, $[\![C]\!]_v = [\![C']\!]_v$ for any v. Therefore $[\![W]\!]_v = [\![B]\!]_v \cap [\![C]\!]_v = [\![B']\!]_v \cap [\![C']\!]_v = [\![W']\!]_v$.
- $W \equiv \neg B$. Then $W' \equiv \neg B'$ for some $B' \equiv_\alpha B$. By the induction hypothesis, $[\![B]\!]_v = [\![B']\!]_v$ for any v. Therefore $[\![W]\!]_v = \mathcal{P} - [\![B]\!]_v = \mathcal{P} - [\![B']\!]_v = [\![W']\!]_v$.
- $W \equiv n \circledR B$. Then $W' \equiv n \circledR B'$ for some $B' \equiv_\alpha B$. By the induction hypothesis, $[\![B]\!]_v = [\![B']\!]_v$ for any v. Therefore $[\![W]\!]_v = \{ P \mid P \equiv (\nu n)Q \text{ for some } Q \in [\![B]\!]_v \} = \{ P \mid P \equiv (\nu n)Q \text{ for some } Q \in [\![B']\!]_v \} = [\![W']\!]_v$.
- $W \equiv \boxdot B$. Then $W' \equiv \boxdot B'$ for some $B' \equiv_\alpha B$. By the induction hypothesis, $[\![B']\!]_v = [\![B]\!]_v$. Therefore $[\![W']\!]_v = \{ P \mid P \to Q \text{ implies } Q \in [\![B']\!]_v \} = \{ P \mid P \to Q \text{ implies } Q \in [\![B]\!]_v \} = [\![W]\!]_v$.
- $W \equiv \forall x.B$. Then $W' \equiv \forall y.B'$ for some $B' \equiv_\alpha B[y/x]$. By the induction hypothesis, $[\![B']\!]_v = [\![B[y/x]]\!]_v$ for any v. Therefore $[\![W']\!]_v = \bigcap_n [\![B'[n/y]]\!]_v = \bigcap_n [\![B[y/x][n/y]]\!]_v = \bigcap_n [\![B[n/x]]\!]_v = [\![W]\!]_v$.
- $W \equiv \text{И}x.B$. Then $fn(W) = fn(W')$ and $W' \equiv \text{И}y.B'$ for some $B' \equiv_\alpha B[y/x]$. By the induction hypothesis, $[\![B']\!]_v = [\![B[y/x]]\!]_v$ for any v. Therefore

$$\begin{array}{rcl} [\![W']\!]_v & = & \bigcup_{n \notin fn(W')} \{ P \mid P \in [\![B'[n/y]]\!]_v \text{ and } n \notin fn(P) \} \\ & = & \bigcup_{n \notin fn(W)} \{ P \mid P \in [\![B[y/x][n/y]]\!]_v \text{ and } n \notin fn(P) \} \\ & = & \bigcup_{n \notin fn(W)} \{ P \mid P \in [\![B[n/x]]\!]_v \text{ and } n \notin fn(P) \} \\ & = & [\![W]\!]_v \end{array}$$

- $W \equiv G(\overline{n})$. Then $W' \equiv G'(\overline{u})$ for some $G' \equiv_\alpha G$. By the induction hypothesis, $[\![G]\!]_v = [\![G']\!]_v$. Therefore $[\![W]\!]_v = [\![G]\!]_v(\overline{n}) = [\![G']\!]_v(\overline{n}) = [\![W']\!]_v$.
- $W \equiv X$. Immediate.
- $W \equiv (\overline{x})B$. Then $W' \equiv (\overline{y})B'$ for some $B' \equiv_\alpha B[\overline{y}/\overline{x}]$. By the induction hypothesis, $[\![B']\!]_v = [\![B[\overline{y}/\overline{x}]]\!]_v$. Therefore $[\![W']\!]_v = \lambda\overline{n}.[\![B'[\overline{n}/\overline{y}]]\!]_v = \lambda\overline{n}.[\![B[\overline{y}/\overline{x}][\overline{n}/\overline{y}]]\!]_v = \lambda\overline{n}.[\![B[\overline{n}/\overline{x}]]\!]_v = [\![W]\!]_v$.
- $W \equiv \nu X.G$. Then $W' \equiv \nu Y G'$ for some $G' \equiv_\alpha G[Y/X]$. By the induction hypothesis, $[\![G']\!]_v = [\![G[Y/X]]\!]_v$ for any v. In particular, for any f, $[\![G']\!]_{v[Y \mapsto f]} = [\![G[Y/X]]\!]_{v[Y \mapsto f]} = [\![G]\!]_{v[X \mapsto f]}$. Therefore $[\![W']\!]_v = \bigsqcup\{ f \mid f \sqsubseteq [\![G']\!]_{v[Y \mapsto f]} \} = \bigsqcup\{ f \mid f \sqsubseteq [\![G]\!]_{v[X \mapsto f]} \} = [\![W]\!]_v$.
- The other cases are similar.

$\square$

Theorem 3.2

1. Suppose X occurs positively in A and F. Then $f \sqsubseteq g$ implies
   (a) $[\![A]\!]_{v[X \mapsto f]} \subseteq [\![A]\!]_{v[X \mapsto g]}$
   (b) $[\![F]\!]_{v[X \mapsto f]} \sqsubseteq [\![F]\!]_{v[X \mapsto g]}$
2. Suppose X occurs negatively in A and F. Then $f \sqsubseteq g$ implies
   (a) $[\![A]\!]_{v[X \mapsto g]} \subseteq [\![A]\!]_{v[X \mapsto f]}$
   (b) $[\![F]\!]_{v[X \mapsto g]} \sqsubseteq [\![F]\!]_{v[X \mapsto f]}$

Proof: By mutual induction on the structures of A and F.

- $A \equiv \mathsf{T}$. Trivial.
- $A \equiv B \wedge C$. Then X occurs positively in B and C.
  1. Assume $f \sqsubseteq g$ and let $P \in [\![A]\!]_{v[X \mapsto f]}$. Then $P \in [\![B]\!]_{v[X \mapsto f]} \cap [\![C]\!]_{v[X \mapsto f]}$. By induction hypothesis *1(a)*, $P \in [\![B]\!]_{v[X \mapsto g]}$ and $P \in [\![C]\!]_{v[X \mapsto g]}$. Therefore $P \in [\![A]\!]_{v[X \mapsto g]}$.
  2. Symmetrical.
- $A \equiv \neg B$. Then X occurs negatively in B
  1. Assume $f \sqsubseteq g$ and let $P \in [\![A]\!]_{v[X \mapsto f]}$. Then $P \in \mathcal{P} - [\![B]\!]_{v[X \mapsto f]}$. By induction hypothesis *2(a)*, $P \in \mathcal{P} - [\![B]\!]_{v[X \mapsto g]}$. Hence $P \in [\![A]\!]_{v[X \mapsto g]}$.
  2. Symmetrical.
- $A \equiv \mathbf{0}$. Trivial.
- $A \equiv B \mid C$. Then X occurs positively in B and C.
  1. Assume $f \sqsubseteq g$ and let $P \in [\![A]\!]_{v[X \mapsto f]}$. Then $P \equiv Q \mid R$ for some $Q \in [\![B]\!]_{v[X \mapsto f]}$ and $R \in [\![C]\!]_{v[X \mapsto f]}$. By induction hypothesis *1(a)*, $Q \in [\![B]\!]_{v[X \mapsto g]}$ and $R \in [\![C]\!]_{v[X \mapsto g]}$. Therefore $P \in [\![A]\!]_{v[X \mapsto g]}$.
  2. Symmetrical.
- $A \equiv B \triangleright C$. Then X occurs positively in C and negatively in B.
  1. Assume $f \sqsubseteq g$ and let $P \in [\![A]\!]_{v[X \mapsto f]}$. Suppose $Q \in [\![B]\!]_{v[X \mapsto g]}$. By induction hypothesis *1(b)*, $Q \in [\![B]\!]_{v[X \mapsto f]}$. Hence $Q \mid P \in [\![C]\!]_{v[X \mapsto f]}$ By induction hypothesis *1(a)*, $Q \mid P \in [\![C]\!]_{v[X \mapsto g]}$. Therefore $P \in [\![A]\!]_{v[X \mapsto g]}$.
  2. Symmetrical.
- $A \equiv n[B]$. Then X occurs positively in B.

1. Assume $f \sqsubseteq g$ and let $P \in [\![A]\!]_{v[X\mapsto f]}$. Then $P \equiv n[Q]$ for some $Q \in [\![B]\!]_{v[X\mapsto f]}$. By induction hypothesis *1(a)*, $Q \in [\![B]\!]_{v[X\mapsto g]}$. Hence $P \in [\![A]\!]_{v[X\mapsto g]}$.
2. Symmetrical.

- $A \equiv B@n$. Then X occurs positively in B.

1. Assume $f \sqsubseteq g$ and let $P \in [\![A]\!]_{v[X\mapsto f]}$. Then $n[P] \in [\![B]\!]_{v[X\mapsto f]}$. By induction hypothesis *1(a)*, $n[P] \in [\![B]\!]_{v[X\mapsto g]}$. Hence $P \in [\![A]\!]_{v[X\mapsto g]}$.
2. Symmetrical.

- $A \equiv n®B$. Then X occurs positively in B.

1. Assume $f \sqsubseteq g$ and let $P \in [\![A]\!]_{v[X\mapsto f]}$. Then $P \equiv (\nu n)Q$ for some $Q \in [\![B]\!]_{v[X\mapsto f]}$. By induction hypothesis *1(a)*, $Q \in [\![B]\!]_{v[X\mapsto g]}$. Hence $P \in [\![A]\!]_{v[X\mapsto g]}$.
2. Symmetrical.

- $A \equiv B \oslash n$. Then X occurs positively in B.

1. Assume $f \sqsubseteq g$ and let $P \in [\![A]\!]_{v[X\mapsto f]}$. Then $(\nu n)P \in [\![B]\!]_{v[X\mapsto f]}$. By induction hypothesis *1(a)*, $(\nu n)P \in [\![B]\!]_{v[X\mapsto g]}$. Hence $P \in [\![A]\!]_{v[X\mapsto g]}$.
2. Symmetrical.

- $A \equiv \boxdot B$. Then X occurs positively in B.

1. Assume $f \sqsubseteq g$ and let $P \in [\![A]\!]_{v[X\mapsto f]}$. Then for any Q such that $P \to Q$ we have $Q \in [\![B]\!]_{v[X\mapsto f]}$. By induction hypothesis *1(a)*, $Q \in [\![B]\!]_{v[X\mapsto g]}$. Hence $P \in [\![A]\!]_{v[X\mapsto g]}$.
2. Symmetrical.

- $A \equiv \forall x.B$. Then X occurs positively in B.

1. Assume $f \sqsubseteq g$ and let $P \in [\![A]\!]_{v[X\mapsto f]}$. Then $P \in [\![B]\!]_{v[X\mapsto f]}$ for any n. By induction hypothesis *1(a)*, $P \in [\![B]\!]_{v[X\mapsto g]}$. Hence $P \in [\![A]\!]_{v[X\mapsto g]}$.
2. Symmetrical.

- $A \equiv \text{И}x.B$. Then X occurs positively in B.

1. Assume $f \sqsubseteq g$ and let $P \in [\![A]\!]_{v[X\mapsto f]}$. Then for some $n \notin fn(A) \cup fn(P)$, $P \in [\![B]\!]_{v[X\mapsto f]}$. By induction hypothesis *1(a)*, $P \in [\![B]\!]_{v[X\mapsto g]}$. Hence $P \in [\![A]\!]_{v[X\mapsto g]}$.
2. Symmetrical.

- $A \equiv F(\overline{n})$. Then X occurs positively in F.

1. Assume $f \sqsubseteq g$. By Induction hypothesis *1(b)*, $[\![F]\!]_{v[X\mapsto f]} \sqsubseteq [\![F]\!]_{v[X\mapsto g]}$. Hence $[\![A]\!]_{v[X\mapsto f]} = [\![F]\!]_{v[X\mapsto f]}(\overline{n}) \subseteq [\![F]\!]_{v[X\mapsto g]}(\overline{n}) = [\![A]\!]_{v[X\mapsto g]}$
2. Symmetrical.

- $F \equiv X$. Immediate.
- $F \equiv (\overline{x})A$. Then X occurs positively in A.
  1. Assume $f \sqsubseteq g$. For any $\overline{n}$,

$$\begin{array}{rcll} [\![F]\!]_{v[X\mapsto f]}(\overline{n}) & = & [\![A[\overline{n}/\overline{x}]]\!]_{v[X\mapsto f]} & \\ & \subseteq & [\![A[\overline{n}/\overline{x}]]\!]_{v[X\mapsto g]} & \text{(induction hypothesis } \mathit{1(a)}\text{)} \\ & = & [\![F]\!]_{v[X\mapsto g]}(\overline{n}) & \end{array}$$

     Hence $[\![F]\!]_{v[X\mapsto f]} \sqsubseteq [\![F]\!]_{v[X\mapsto g]}$.
  2. Symmetrical.
- $F \equiv \nu Z.G$. Then Z occurs positively in G.
  1. Assume $f \sqsubseteq g$. If $Z = X$ then the result is immediate. Now suppose $Z \neq X$. By definition,

$$\begin{array}{l} [\![F]\!]_{v[X\mapsto f]} = \bigsqcup\{\, h \mid h \sqsubseteq [\![G]\!]_{v[X\mapsto f][Z\mapsto h]} \,\} \\ [\![F]\!]_{v[X\mapsto g]} = \bigsqcup\{\, h \mid h \sqsubseteq [\![G]\!]_{v[X\mapsto g][Z\mapsto h]} \,\} \end{array}$$

     For any $\overline{n}$, let $P \in [\![F]\!]_{v[X\mapsto f]}(\overline{n})$, *i.e.* $P \in h(\overline{n})$ for some h s.t. $h \sqsubseteq [\![G]\!]_{v[X\mapsto f][Z\mapsto h]}$. By induction hypothesis *1(b)*,

$$\begin{array}{rcl} [\![G]\!]_{v[X\mapsto f][Z\mapsto h]} & = & [\![G]\!]_{(v[X\mapsto h])[Z\mapsto f]} \\ & \sqsubseteq & [\![G]\!]_{(v[X\mapsto h])[Z\mapsto g]} \\ & = & [\![G]\!]_{(v[X\mapsto g])[Z\mapsto h]} \end{array}$$

     Hence $h \sqsubseteq [\![G]\!]_{v[X\mapsto g][Z\mapsto h]}$. Therefore $P \in [\![F]\!]_{v[X\mapsto g]}(\overline{n})$. It follows that $[\![F]\!]_{v[X\mapsto f]}(\overline{n}) \subseteq [\![F]\!]_{v[X\mapsto g]}(\overline{n})$. Since $\overline{n}$ is arbitrary, we conclude $[\![F]\!]_{v[X\mapsto f]} \sqsubseteq [\![F]\!]_{v[X\mapsto g]}$.
  2. Symmetrical.

This completes the induction. $\square$

Lemma A.1 *Given a transposition τ and a function $f : \overline{\mathcal{N}} \to 2^{\mathcal{P}}$, define $f^\tau : \overline{\mathcal{N}} \to 2^{\mathcal{P}}$ thus: $f^\tau(\overline{n}) = f(\overline{n}) \cup (f(\overline{n}\tau))\tau$ for any $\overline{n}$. Then*

1. *τ is preserved by f^τ.*
2. *If $f \sqsubseteq g$ and τ is preserved by g then $f^\tau \sqsubseteq g$.*

Proof:

1. For ant $\overline{n}$,

$$\begin{array}{rl} (f^\tau(\overline{n}))\tau & = (f(\overline{n}))\tau \cup ((f(\overline{n}\tau))\tau)\tau \\ & = (f((\overline{n}\tau)\tau))\tau \cup f(\overline{n}\tau) \\ & = f^\tau(\overline{n}\tau) \end{array}$$

2. For any $\overline{n}$, since $f \sqsubseteq g$, $f(\overline{n}) \subseteq g(\overline{n})$ and $f(\overline{n}\tau) \subseteq g(\overline{n}\tau)$. since τ is preserved by g, $(g(\overline{n}\tau))\tau = g(\overline{n}\tau\tau) = g(\overline{n})$. Hence

$$\begin{array}{rcl} f^{\tau}(\overline{n}) & = & f(\overline{n}) \cup (f(\overline{n}\tau))\tau \\ & \subseteq & g(\overline{n}) \cup (g(\overline{n}\tau))\tau \\ & = & g(\overline{n}) \cup g(\overline{n}) \\ & = & g(\overline{n}) \end{array}$$

□

Proposition 3.5

1. If τ is preserved by v on *fpv*(A) then $([\![A]\!]_v)\tau = [\![A\tau]\!]_v$.
2. If τ is preserved by v on *fpv*(F) then τ is preserved by $[\![F]\!]_v$.

Proof: By mutual induction on the structures of A and F. Instead of the equality in *1*, we shall prove $([\![A]\!]_v)\tau \subseteq [\![A\tau]\!]_v$. The other direction of the inclusion will follow immediately: If $P \in [\![A\tau]\!]_v$, then $P = P\tau\tau \in ([\![A\tau]\!]_v)\tau\tau \subseteq ([\![A\tau\tau]\!]_v)\tau = ([\![A]\!]_v)\tau$. For the same reason it suffices to prove $([\![F]\!]_v(\overline{n}))\tau \subseteq [\![F]\!]_v(\overline{n}\tau)$ for any $\overline{n}$, in *2*.

- $A \equiv \mathsf{T}$. Trivial.
- $A \equiv B \wedge C$. Let $P \in ([\![A]\!]_v)\tau$. Then $P\tau \in [\![A]\!]_v$, *i.e.* $P\tau \in [\![B]\!]_v \cap [\![C]\!]_v$. Hence $P \in [\![B]\!]_v\tau$ and $P\tau \in [\![C]\!]_v\tau$. By induction hypothesis *1*, $P \in [\![B\tau]\!]_v$ and $P\tau \in [\![C\tau]\!]_v$. Therefore $P \in [\![A\tau]\!]_v$.
- $A \equiv \mathbf{0}$. Trivial.
- $A \equiv n[B]$. Let $P \in ([\![A]\!]_v)\tau$. Then $P\tau \in [\![A]\!]_v$, *i.e.* $P\tau \equiv n[Q]$ for some Q such that $Q \in [\![B]\!]_v$. Hence $Q\tau \in [\![B]\!]_v\tau$. By induction hypothesis *1*, $Q\tau \in [\![B\tau]\!]_v$. Since $A\tau \equiv n\tau[B\tau]$ and $P \equiv P\tau\tau \equiv n\tau[Q\tau]$, we conclude $P \in [\![A\tau]\!]_v$.
- $A \equiv B \mid C$. Let $P \in ([\![A]\!]_v)\tau$. Then $P\tau \in [\![A]\!]_v$, *i.e.* $P\tau \equiv Q \mid R$ for some Q and R such that $Q \in [\![B]\!]_v$ and $R \in [\![C]\!]_v$. By induction hypothesis *1*, $Q \in [\![B\tau]\!]_v$ and $R \in [\![C\tau]\!]_v$. Hence $P \equiv Q\tau \mid R\tau \in [\![B\tau \mid C\tau]\!]_v = [\![A\tau]\!]_v$.
- $A \equiv B \rhd C$. Let $P \in ([\![A]\!]_v)\tau$. Then $P\tau \in [\![A]\!]_v$, *i.e.* for any $Q \in [\![B]\!]_v$, $Q \mid P\tau \in [\![C]\!]_v$. Let $Q' \in [\![B\tau]\!]_v$. Then, by induction hypothesis *1*, $Q' \in ([\![B]\!]_v)\tau$. Hence $Q'\tau \in [\![B]\!]_v$. Therefore $(Q' \mid P)\tau = Q'\tau \mid P\tau \in [\![C]\!]_v$. So $Q' \mid P \in [\![C]\!]_v\tau$. By induction hypothesis *1* again, $Q' \mid P \in [\![C\tau]\!]_v$. It follows that $P \in [\![B\tau \mid C\tau]\!]_v = [\![A\tau]\!]_v$.
- $A \equiv n®B$. Let $P \in ([\![A]\!]_v)\tau$. Then $P\tau \in [\![A]\!]_v$, *i.e.* $P\tau \equiv (\nu n)Q$ for some $Q \in [\![B]\!]_v$. So $Q\tau \in ([\![B]\!]_v)\tau$. By induction hypothesis *1*, $Q\tau \in [\![B\tau]\!]_v$. Hence $P \equiv ((\nu n)Q)\tau = (\nu(n\tau))(Q\tau) \in [\![(\nu(n\tau))(B\tau)]\!]_v = [\![A\tau]\!]_v$.
- $A \equiv B \oslash n$. Let $P \in ([\![A]\!]_v)\tau$. Then $P\tau \in [\![A]\!]_v$, *i.e.* $(\nu n)P\tau \in [\![B]\!]_v$. Hence $((\nu n)P\tau)\tau \in [\![B]\!]_v\tau$. By induction hypothesis *1*, $(\nu(n\tau))P = ((\nu n)P\tau)\tau \in [\![B\tau]\!]_v$. Therefore $P \in [\![(B\tau) \oslash (n\tau)]\!]_v = [\![A\tau]\!]_v$.

- $A \equiv \boxdot B$. Let $P \in ([\![A]\!]_v)\tau$. Then $P\tau \in [\![\boxdot B]\!]_v$. So for any Q such that $P\tau \to Q$ we have $Q \in [\![B]\!]_v$. By Proposition 2.1, $P = (P\tau)\tau \to Q\tau$. By induction hypothesis *1*, $Q\tau \in ([\![B]\!]_v)\tau$. Hence $P \in [\![\boxdot(B\tau)]\!]_v = [\![(\boxdot B)\tau]\!]_v$.

- $A \equiv \forall x.B$. Let $P \in ([\![A]\!]_v)\tau$. Then $P\tau \in [\![\forall x.B]\!]_v$. Hence $P\tau \in [\![B[k/x]]\!]_v$, or equally, $P \in ([\![B[k/x]]\!]_v)\tau$, for any k. For any n, let $m = n\tau$. Then $n = m\tau$. By induction hypothesis *1*, $P \in [\![(B[m/x])\tau]\!]_v = [\![(B\tau)[m\tau/x]]\!]_v = [\![(B\tau)[n/x]]\!]_v$. Hence $P \in [\![\forall x.(B\tau)]\!]_v = [\![(\forall x.B)\tau]\!]_v = [\![A\tau]\!]_v$.

- $A \equiv \text{И}x.B$. Then $A\tau \equiv \text{И}x.(B\tau)$. Let $\tau = [m \leftrightarrow n]$ and $P \in ([\![A]\!]_v)\tau$. Write Q for $P\tau$. Then $P \equiv Q\tau$ and $Q \in [\![A]\!]_v$. Hence $Q \in [\![B[k/x]]\!]_v$ for some $k \notin fn(A) \cup fn(Q)$. Let l be a name such that $l \notin \{m, n, k\} \cup fn(A) \cup fn(P)$. Then $l \notin fn(Q) \cup fn(B[k/x])$. Write τ' for $[k \leftrightarrow l]$. Since $k, l \notin fn(P\tau)$, $P\tau = (P\tau)\tau' = Q\tau' \in ([\![B[k/x]]\!]_v)\tau'$. By induction hypothesis *1*, $([\![B[k/x]]\!]_v)\tau' \subseteq [\![(B[k/x])\tau']\!]_v = [\![B[l/x]]\!]_v$. Hence $P = (P\tau)\tau \in ([\![B[l/x]]\!]_v)\tau$. By induction hypothesis *1* again,

$$P \in ([\![B[l/x]]\!]_v)\tau \subseteq [\![(B[l/x])\tau]\!]_v = [\![(B\tau)[l/x]]\!]_v$$

  Since $l \notin fn(\text{И}x.B) \cup fn(P)$, we conclude $P \in [\![\text{И}x.B]\!]_v \equiv [\![A]\!]_v$.

- $A \equiv F(\overline{n})$. Then $A\tau = F(\overline{n}\tau)$. By induction hypothesis *2*, τ is preserved by $[\![F]\!]_v$ on $fpv(F)$. So $([\![F]\!]_v(\overline{n})\tau = [\![F]\!]_v(\overline{n}\tau)$. Therefore

$$([\![A]\!]_v)\tau = ([\![F]\!]_v(\overline{n}))\tau = [\![F]\!]_v(\overline{n}\tau) = [\![F(\overline{n}\tau)]\!]_v = [\![A\tau]\!]_v$$

- $F \equiv X$. Trivial.

- $F \equiv (\overline{x})A$. For any $\overline{n}$,

$$\begin{array}{rcll} [\![F(\overline{n})]\!]_v & = & [\![A[\overline{n}/\overline{x}]]\!]_v\tau & \\ & = & [\![(A[\overline{n}/\overline{x}])\tau]\!]_v & \text{Induction hypothesis } \mathit{1} \\ & = & [\![A[\overline{n}\tau/\overline{x}]]\!]_v & \text{because } fn(A) = \emptyset \text{ and } fv(A) \subseteq \{\overline{x}\} \\ & = & [\![F(\overline{n}\tau)]\!]_v & \end{array}$$

- $F \equiv \nu Z.G$. Then Z occurs positively in G. To show $([\![F]\!]_v(\overline{n}))\tau \subseteq [\![F]\!]_v(\overline{n}\tau)$ for any $\overline{n}$, let $P \in ([\![F]\!]_v(\overline{n}))\tau$. Then $P \in f(\overline{n})$ for some f such that $f \sqsubseteq [\![G]\!]_{v[Z \mapsto f]}$ and $P \in (f(\overline{n}))\tau$. Therefore there is $Q \in f(\overline{n})$ such that $P \equiv Q\tau$. Let f^τ be as defined in Lemma A.1. Then $f \sqsubseteq f^\tau$ and τ is preserved by f^τ. By Theorem 3.2, $f \sqsubseteq [\![G]\!]_{v[Z \mapsto f]} \sqsubseteq [\![G]\!]_{v[Z \mapsto f^\tau]}$. Since τ is preserved by $v[Z \mapsto f^\tau]$ on $fpv(G)$, τ is preserved by $[\![G]\!]_{v[Z \mapsto f^\tau]}$ by induction hypothesis *2*. By Lemma A.1, $f^\tau \sqsubseteq [\![G]\!]_{v[Z \mapsto f^\tau]}$. Now $P = Q\tau \in (f(\overline{n}))\tau \subseteq (f^\tau(\overline{n}))\tau = f^\tau(\overline{n}\tau)$. Hence $P \in [\![F(\overline{n}\tau)]\!]_v$.

This completes the proof. □