Departmental Seminar
Oxford University Computing Laboratory
18 November 2003

# Reasoning with Names

Ian Stark

Laboratory for Foundations of Computer Science
School of Informatics
University of Edinburgh

# What's in a name?

The idea of a *name* arises repeatedly across computer science, as an abstract piece of data that carries identity but little else. Typically, names can be compared with each other, and there is an unlimited supply of fresh names, but that is all.

Names are useful, convenient, and often very comfortable to reason about informally, but turn out to be tremendously slippery in formal reasoning.

# Some uses of names in computer science

**Programming:** local variables; procedure parameters; $\lambda x.M$; $\alpha$-conversion.

**Logic:** quantifiers $\forall x.\phi$, $\exists y.P$.

**Objects:** identity; references; pointers.

**Security:** nonces; privacy; authentication.

**Communication:** channels, TCP/IP sockets, thread IDs, $\pi$-calculus $(\nu x)P$.

**Distributed systems:** locations, namespaces.

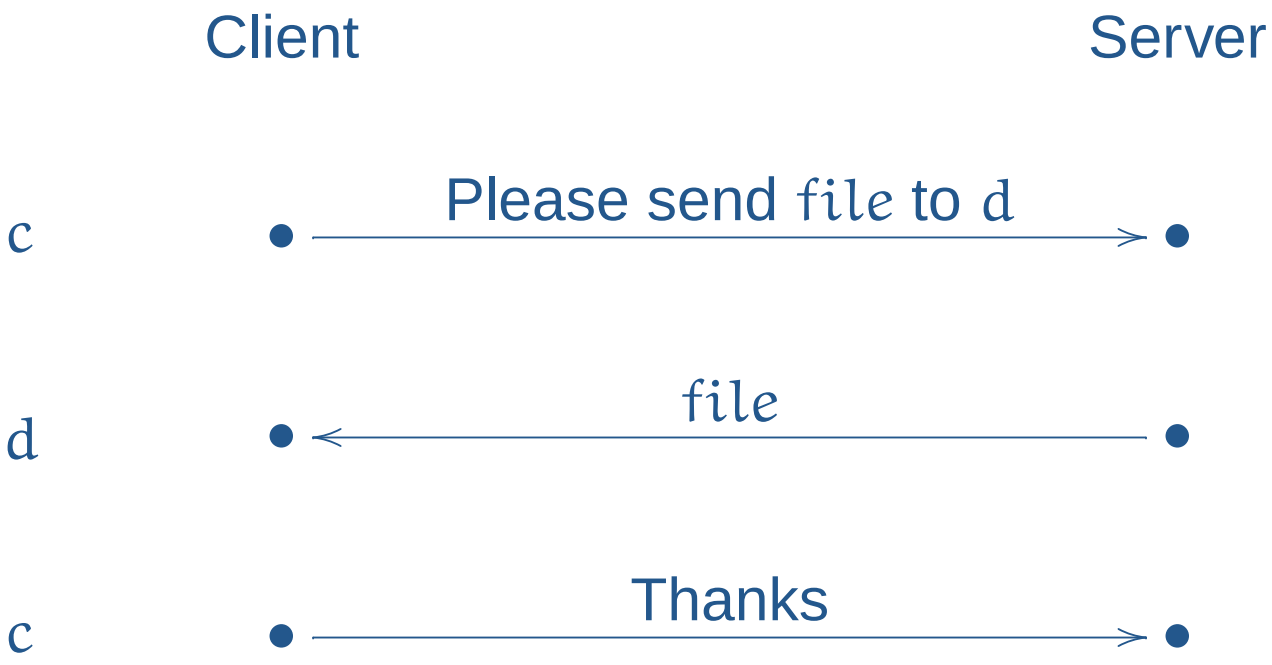
# Overview of talk

- Some uses of names and naming
- FTP and the local area $\pi$-calculus
- Models for local names based on varying sets
- Metalogics and mechanised reasoning for names
- FM-sets, FreshML and nominal logic: $И\mathfrak{a}.\phi(\mathfrak{x}, \mathfrak{a})$

# FTP: File Transfer Protocol

# FTP: File Transfer Protocol

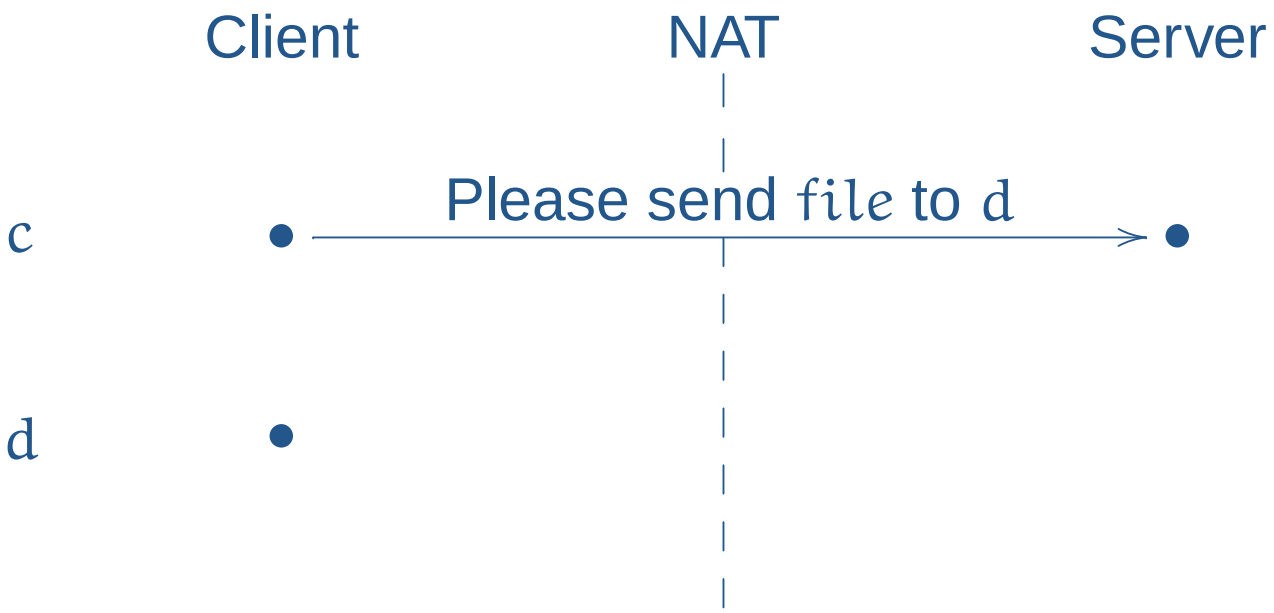
# FTP: File Transfer Protocol



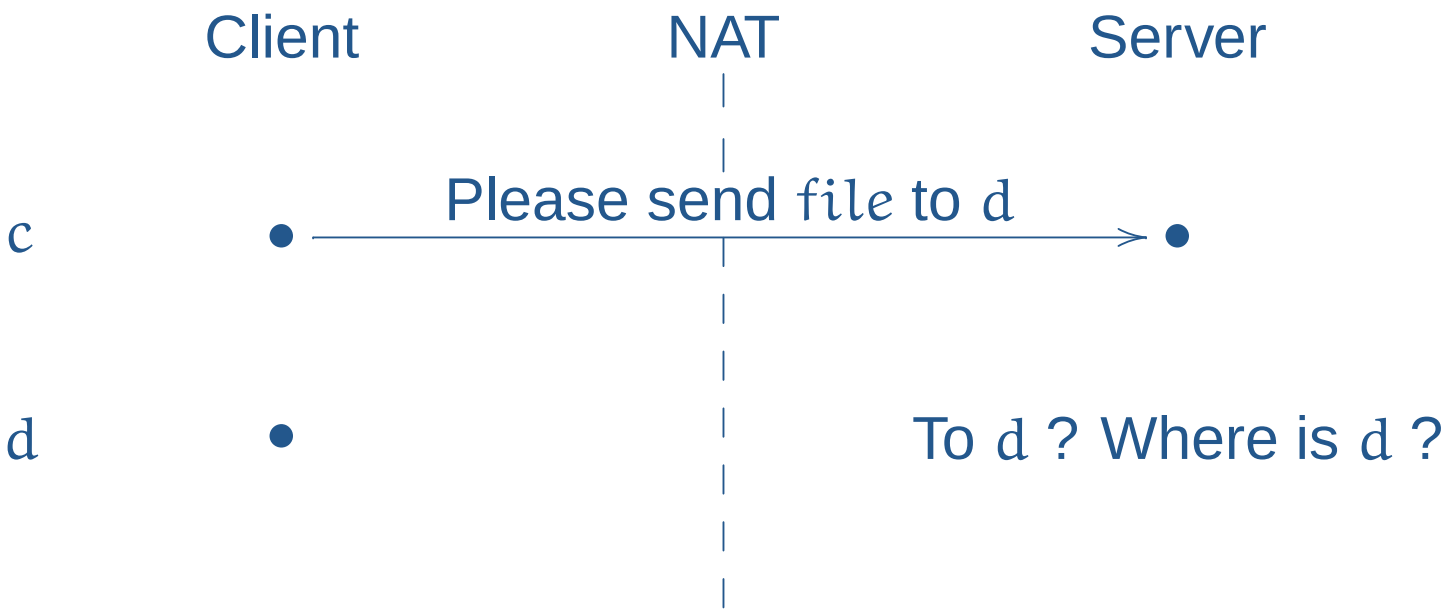
# FTP: File Transfer Protocol


The channel identifier $d$ is a name, and we can give a natural interpretation of FTP in systems like the $\pi$-calculus that support name-passing processes.

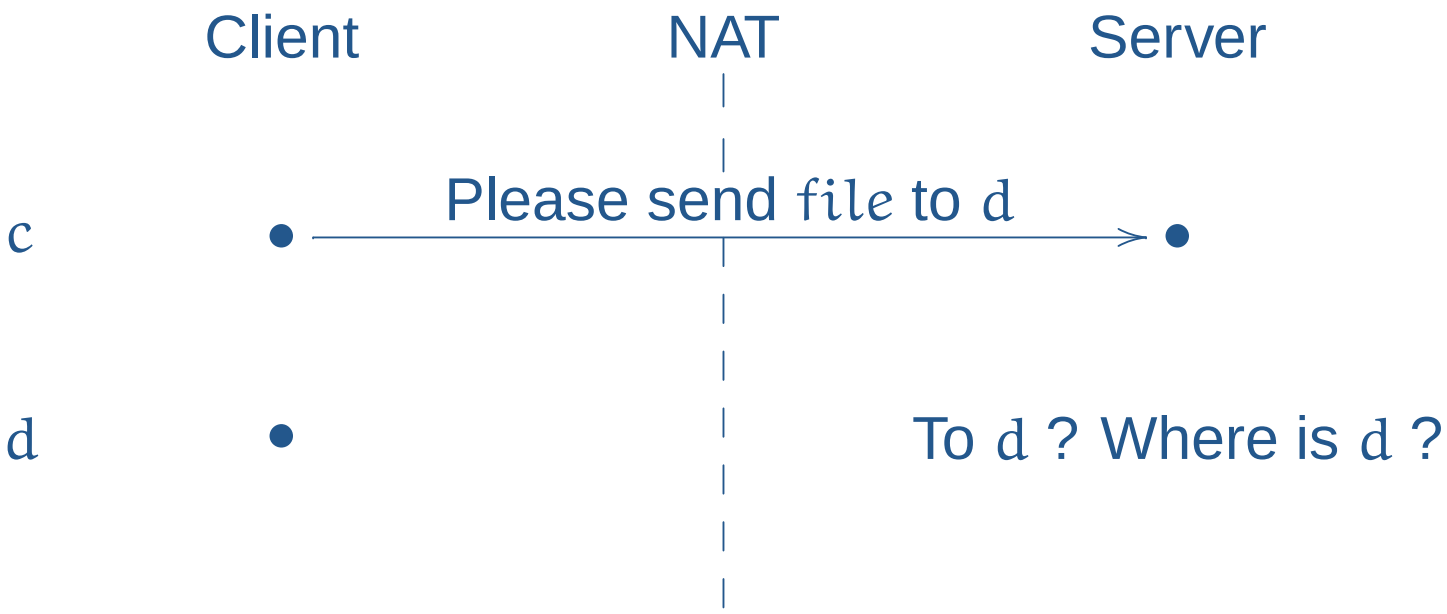
# FTP and Network Address Translation



# FTP and Network Address Translation



# FTP and Network Address Translation




Network address translation (NAT) routes data between different name spaces, and so breaks the name-passing used in FTP.

To capture what has gone wrong here, we can look more closely at the nature of the names involved.

# FTP implementation


Numbers like 10.0.0.4 and 21 have specific effects on each local network or machine; but they are also globally "well-known".

To work with this we need to distinguish the scope over which a name is known from the range across which it acts.

*"Think globally, act locally"*

# The local area $\pi$-calculus

(Chothia, Stark)

- Refines the $\pi$-calculus with *local areas* of interaction.
- Channels have *levels* to determine their range of communication.

| | |
|---|---|
| Processes | $net[\, host[\, P \mid Q \,] \mid R \mid host[\, S \,] \,]$ |
| Channels | $a@net;\ b@host;\ u, v@app$ |

- This can model FTP failing over NAT (and how 'passive' FTP succeeds)
- Also Napster's peer-to-peer interaction across firewalls.

Other settings where this is relevant include standard libraries, mobile agents and service discovery.

(Netgear and U. Wisconsin)

# Names in many places

Often the interest is not in names themselves, but in how they interact with other features. For example:

- Names and communicating processes. (join-, $\pi$-calculus)
- Higher-order functions with names: $\nu n.\lambda f.fn$ .
  (nu-calculus, Pitts+Stark)
- Names as encryption keys. (spi calculus, Abadi+Gordon)
- Naming mobile locations. (ambients, Cardelli+Gordon)
- Local names within semistructured data.
  (trees with hidden labels, Gardner/Ghelli/Cardelli)

# What's the difficulty?

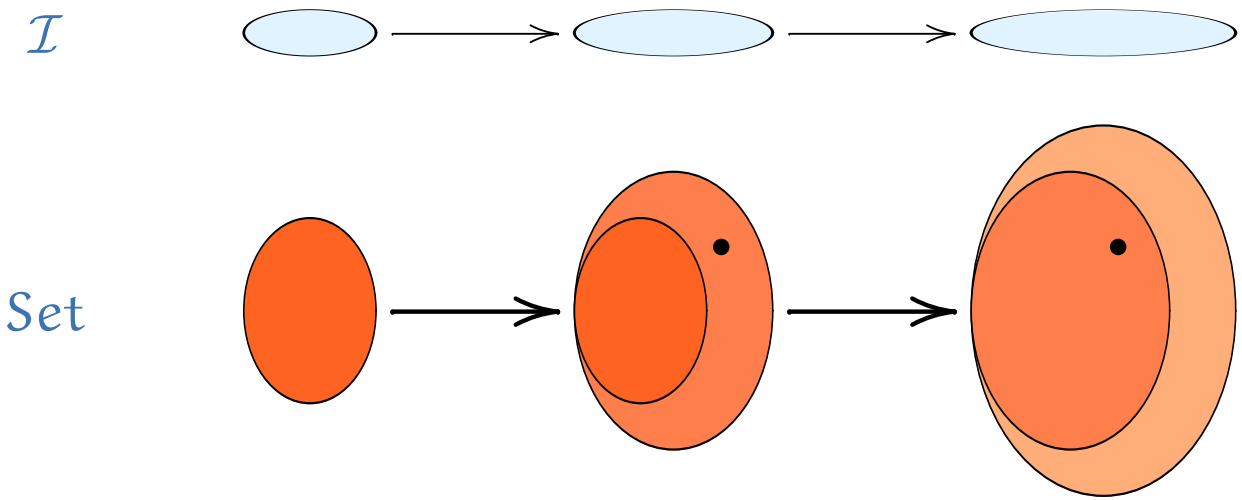
Concrete implementation of names requires care, but is generally manageable: integers, addresses, some choice of globally unique ID.

Informal reasoning is also fairly natural: be aware of aliasing, keep names distinct, and everything will be OK.

Yet to make this formal, or to mechanise reasoning about names, turns out to be surprisingly hard.

# Models for names

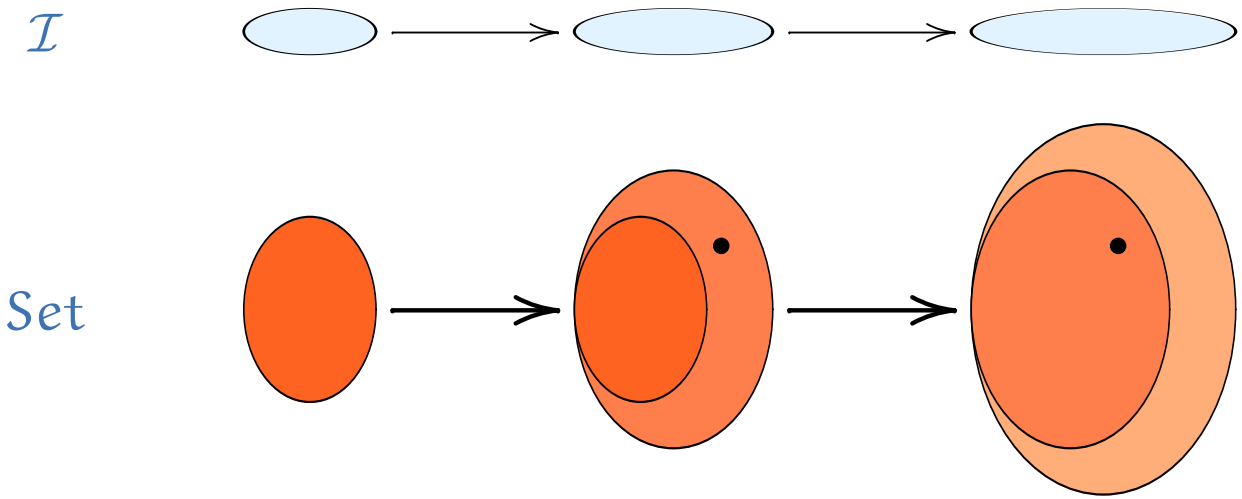
We can build a denotational semantics that accounts for names by using structures that vary according to the names available.




A *varying set* $\mathrm{B} \in Set^{\mathcal{I}}$ specifies for any finite set of names $s$ the set $\mathrm{B}(s)$ of values using names from $s$, together with information about how these values change under renaming.

# Structure within $\mathcal{S}et^{\mathcal{I}}$

Varying sets offer lots of convenient structure to work with, while keeping us honest about the impact of names.




- Pairs $\mathrm{A} \times \mathrm{B}$ and function space $\mathrm{A} \to \mathrm{B}$.
- Separated pairs $\mathrm{A} \otimes \mathrm{B}$ and fresh function space $\mathrm{A} \multimap \mathrm{B}$.
- The varying set of names $\mathrm{N}$ and its function space $\mathrm{N} \multimap \mathrm{A}$.

# Models in varying sets

$\mathcal{S}et^{\mathcal{I}}$ has proved a fruitful setting for models of naming.

- Denotational semantics for the nu-calculus.
- Mutable store and pointers in *Reduced ML*.
- Full abstraction for the $\pi$-calculus ($\geq 4$ times).
- Free algebras with enriched arities for all of the above.

Other choices of base and index are also useful:

- $\mathcal{C}\mathrm{po}^{\mathcal{I}}$ for recursive programs or processes.
- $\mathcal{S}et^{\mathcal{S}}$ for local state in Algol.
- $\mathcal{S}et^{\mathcal{V}}$ for abstract syntax with binders.

# Reasoning about names

A sound and adequate model gives a valid reasoning method, but it can be hard work. Other methods include:

- *Logical relations* between name sets or state sets
  e.g. proving correctness of a memoisation operator.
- *Separation logic* for heaps and pointers; $\phi * \psi$, $\phi \mathrel{-\!\!*} \psi$
  e.g. in-place list reversal, graph marking.
- *Bunched implications* for all kinds of resources
  e.g. $\phi * \psi$, $\phi \wedge \psi$, $\forall_{new} x.\phi(x)$.

A further generalisation is to look for *metalogics* that provide support for reasoning about names and binding.

# Working with binders

Suppose that we write a program to manipulate $\lambda$-calculus terms.

$$\begin{array}{lll} \mathtt{datatype}\ \mathtt{Term} = \mathtt{var}\ \ \mathtt{of}\ \mathtt{Name} & & x \\ \quad | \ \mathtt{app}\ \mathtt{of}\ \mathtt{Term} * \mathtt{Term} & & (MN) \\ \quad | \ \mathtt{lam}\ \mathtt{of}\ \mathbf{Abstn} & & \lambda x.M \end{array}$$

We want to choose $\mathbf{Abstn}$ in a way that gives:

- uniform behaviour under $\alpha$-conversion;
- recursively defined functions on $\mathtt{Term}$;
- proof by induction over the structure of $\mathtt{Term}$.

*"In this situation the common practice of human provers is to say one thing and do another"*

# Some approaches to formalising binding

- Use de Bruijn indices. (drop names entirely)
- Axiomatize what's required of $\mathbf{Abstn}$. (Gordon, Melham)

- $\mathbf{Abstn} = \mathrm{Name} * \mathrm{Term}$. (Pollack, McKinna)
- $\mathbf{Abstn} = \mathrm{Term} \to \mathrm{Term}$. (Higher-Order Abstact Syntax)
- $\mathbf{Abstn} = \mathrm{Name} \to \mathrm{Term}$.

- Fraenkel-Mostowski set theory. (Pitts, Gabbay, Shinwell)

# FM set theory

Originally created to show independence of the Axiom of Choice.

FM-sets can include *atoms* from a countably infinite set $\mathbb{A}$. Permutations on $\mathbb{A}$ then induce permutations on the sets.

All sets must have *finite support*, and every operation on them is *equivariant* under permutation of the underlying names.

We get all the constructions of ZF set theory, together with a new *abstraction* set former $[\mathbb{A}]X$ capturing $\alpha$-conversion. If we take

$$\mathbf{Abstn} = [\mathrm{Name}]\mathrm{Term}$$

and program with this, then recursive definitions and inductive proofs all follow smoothly.

# Fresh ML

www.freshml.org

*"... a new language derived from Standard ML which provides superior facilities for writing software systems which manipulate syntax involving binding operations."*

```
val identity = let fresh x:Name in lam(<x>(var x)) end

fun subst (e, x, var y)       = if y=x then e else var y
  | subst (e, x, lam(<y>e1)) = lam(<y>(subst(e, x, e1)))
  | subst (e, x, app(e1,e2)) =
                    app(subst(e, x, e1), subst(e, x, e2))
```

Internalises $\alpha$-conversion while supporting recursive functions and inductive proofs over the $\mathrm{Term}$ datatype.

(Also now Fresh O'Caml and $\alpha$-Prolog.)

# Nominal logic

A first-order theory of FM sets. Axioms cover *name swapping* $(a\,b)\cdot x$ and *freshness* $a\,\#\,x$, with properties like:

$$a\,\#\,x \;\wedge\; a'\,\#\,x \;\Longrightarrow\; (a\,a')\cdot x = x\ .$$

From these we can define a *freshness quantifier* asserting "for some/any fresh name":

$$И a.\phi(a,\vec{x})$$

This has an introduction rule like $\exists$, and eliminates like $\forall$.

For example, $\eta$-equivalence between $\lambda$-terms can be phrased as

$$\forall t{:}\mathit{Term}\ .\ И a{:}\mathit{Name}\ .\ t = \lambda a.ta\ .$$

# Summary: names are worth taking seriously

- Names give interesting behaviour, in many settings.
- $Set^{\mathcal{I}}$ has rich structure for modelling names ($\times$, $\rightarrow$, $\otimes$, $\multimap$).
- FM set theory and logics of freshness support abstract reasoning about names.

## Some active areas

- Modal logics for names in processes
- Spatial reasoning; pointers in data
- A fully-abstract model for the nu-calculus
- Higher-order nominal logic; FM type theory.
- Bringing more powerful tools like logical relations into the metatheory.

... as well as applying all these techniques to the uses of names given right back at the beginning.

# Rules for the freshness quantifier $И a.\phi$

Definition:

$$И a.\phi(a,\vec{x}) \overset{\Delta}{\iff} \exists a.(a \# \vec{x}) \wedge \phi(a,\vec{x})$$
$$\iff \forall a.(a \# \vec{x}) \Rightarrow \phi(a,\vec{x})$$

Natural deduction rules:

$$\frac{\Phi, a \# \vec{x} \vdash \phi}{\Phi \vdash И a.\phi}\ (И - \text{intro}) \qquad \frac{\Phi \vdash И a.\phi \quad \Phi, a \# \vec{x}, \phi \vdash \psi}{\Phi \vdash \psi}\ (И - \text{elim})$$

Sequent calculus rules:

$$\frac{\Phi, a \# \vec{x}, \phi \vdash \Psi}{\Phi, И a.\phi \vdash \Psi}\ (И - \text{left}) \qquad \frac{\Phi, a \# \vec{x} \vdash \phi, \Psi}{\Phi \vdash И a.\phi, \Psi}\ (И - \text{right})$$