Introduction

Meta-Programming
Introduction

Meta-Programming

\[ \begin{align*}
\text{meta-level} \\
\text{object-level}
\end{align*} \]
Introduction

Meta-Programming

\[
\begin{array}{c|c}
\text{meta-level} & \text{PCF} \\
\hline
\text{object-level} & \text{booleans, numbers}
\end{array}
\]
Introduction

Meta-Programming

<table>
<thead>
<tr>
<th>meta-level</th>
<th>???</th>
</tr>
</thead>
<tbody>
<tr>
<td>object-level</td>
<td>booleans, numbers, λ-calculus</td>
</tr>
</tbody>
</table>

Steffen Lösch and Andrew M. Pitts

Full Abstraction for PCF with Names
Introduction

Meta-Programming

<table>
<thead>
<tr>
<th>meta-level</th>
<th>PNA</th>
</tr>
</thead>
<tbody>
<tr>
<td>object-level</td>
<td>booleans, numbers, $\lambda$-calculus</td>
</tr>
</tbody>
</table>
## Introduction

### Meta-Programming

<table>
<thead>
<tr>
<th>meta-level</th>
<th>PNA</th>
</tr>
</thead>
<tbody>
<tr>
<td>object-level</td>
<td>booleans, numbers, $\lambda$-calculus</td>
</tr>
</tbody>
</table>

 operational semantics $\simeq$
Introduction

Meta-Programming

\textit{meta-level} \quad \text{PNA} \quad \textit{object-level}

booleans, numbers, \(\lambda\)-calculus

\text{operational semantics} \overset{\sim}{\Rightarrow} \quad \text{denotational semantics} =
Meta-Programming

**meta-level**

**object-level**

| PNA | booleans, numbers, $\lambda$-calculus |

---

**adequacy**

operational semantics $\simeq$

denotational semantics $=$
Introduction

Meta-Programming

- meta-level
- object-level

PNA
- booleans, numbers, λ-calculus

adequacy

operational semantics \(\simeq\)

denotational semantics \(=\)

full abstraction
Overview

1. Introduction
Overview

1. Introduction

2. PCF
   ▶ language and operational semantics
   ▶ domain theory and denotational semantics for PCF
   ▶ parallel-or and full abstraction
Overview

1. Introduction

2. PCF
   ▶ language and operational semantics
   ▶ domain theory and denotational semantics for PCF
   ▶ parallel-or and full abstraction

3. PCF with names = PNA
   ▶ motivation and language
   ▶ operational semantics
Overview

1. Introduction

2. PCF
   - language and operational semantics
   - domain theory and denotational semantics for PCF
   - parallel-or and full abstraction

3. PCF with names = PNA
   - motivation and language
   - operational semantics

4. Nominal domain theory and full abstraction for PNA
PCF – Syntax

\[\begin{align*}
\text{Types} & \quad \tau ::= \text{bool} \mid \text{nat} \mid \tau \times \tau \mid \tau \tau \\
\text{Expressions} & \quad e ::= T \mid F \mid \text{if } e \text{ then } e \text{ else } e \\
& \quad \text{booleans and if } O \mid S e \mid \text{pred } e \mid \text{zero } e \\
& \quad \text{numbers and zero test } (e, e) \mid \text{fst } e \mid \text{snd } e \\
& \quad \text{pairs and projections } x \mid \lambda x : \tau \ e \mid e e \\
& \quad \text{variable } (x \in V) \text{ and functions } \\
& \quad \text{fix } e \text{ fixed-point recursion }
\end{align*}\]
PCF – Syntax

▶ Types

\[ \tau ::= \text{bool} \mid \text{nat} \mid \tau \times \tau \mid \tau \rightarrow \tau \]

▶ Expressions

\[ e ::= T \mid F \mid \text{if } e \text{ then } e \text{ else } e \mid \text{booleans and if } o \mid S \mid e \mid \text{pred } e \mid \text{zero } e \mid \text{numbers and zero test } (e, e) \mid \text{fst } e \mid \text{snd } e \mid \text{pairs and projections } x \mid \lambda x: \tau \mid e \; e \mid \text{variable } (x \in V) \mid \text{functions } \text{fix } e \]

▶ Binding forms

\[ \lambda x: \tau \]

Steffen Lösch and Andrew M. Pitts  Full Abstraction for PCF with Names
PCF – Syntax

- **Types**
  \[ \tau ::= \text{bool} \mid \text{nat} \mid \tau \times \tau \mid \tau \rightarrow \tau \]

- **Expressions**
  \[ e ::= \]

  - booleans and if
    \[ T \mid F \mid \text{if } e \text{ then } e \text{ else } e \mid \]

  - numbers and zero test
    \[ 0 \mid S e \mid \text{pred } e \mid \text{zero } e \mid \]

  - pairs and projections
    \[ (e, e) \mid \text{fst } e \mid \text{snd } e \mid \]

  - variable (\(x \in \mathbb{V}\)) and functions
    \[ x \mid \lambda x : \tau \rightarrow e \mid e e \mid \text{fix } e \]
PCF – Syntax

▶ Types
\[ \tau ::= \text{bool} \mid \text{nat} \mid \tau \times \tau \mid \tau \to \tau \]

▶ Expressions
\[ e ::= T \mid F \mid \text{if } e \text{ then } e \text{ else } e \mid \text{booleans and if} \]
\[ 0 \mid S \ e \mid \text{pred } e \mid \text{zero } e \mid \text{numbers and zero test} \]
\[ (e, e) \mid \text{fst } e \mid \text{snd } e \mid \text{pairs and projections} \]
\[ x \mid \lambda x : \tau \to e \mid \text{variable } (x \in \mathbb{V}) \text{ and functions} \]
\[ \text{fix } e \mid \text{fixed-point recursion} \]

▶ Binding forms
\[ \lambda x : \tau \to - \]
PCF – Typing and Examples

Typing judgement: \( \Gamma \vdash e : \tau \) 

\( \Gamma \vdash x : \tau \) 

\( c = T \) | \( F \) 

\( \Gamma \vdash c : bool \) 

\( \Gamma \vdash e : nat \) 

\( \Gamma \vdash \text{zero} e : bool \) 

\( \Gamma, x : \tau \vdash e : \tau' \) 

\( \Gamma \vdash \lambda x : \tau e : \tau \tau' \) 

Currying

\( \lambda f : (\tau_1 \times \tau_2 \tau_3) \lambda x_1 : \tau_1 \lambda x_2 : \tau_2 f(x_1, x_2) \) 

Sum of numbers:

\( x + y \) 

\( \text{fix} (\lambda f : nat \times nat \times nat \lambda x : nat \lambda y : nat \text{if zero } y \text{ then } x \text{ else } S(f x (\text{pred } y))) \)
Typing judgement: $\Gamma \vdash e : \tau$

**Typing judgement:** $\Gamma \vdash e : \tau$
Typing judgement: $\Gamma \vdash e : \tau$

$\Gamma = \{x_1 : \tau_1, \ldots, x_n : \tau_n\}$
Typing judgement: $\Gamma \vdash e : \tau$

$\Gamma = \{x_1 : \tau_1, \ldots, x_n : \tau_n\}$

Typing judgement: $\Gamma \vdash e : \tau$

$\begin{array}{c}
(x : \tau) \in \Gamma \\
\hline 
\Gamma \vdash x : \tau 
\end{array}$
Typing judgement: $\Gamma \vdash e : \tau$

$\Gamma = \{x_1 : \tau_1, \ldots, x_n : \tau_n\}$

- $\Gamma \vdash x : \tau$
- $c = T | F$
- $\Gamma \vdash c : bool$
Typing judgement: $\Gamma \vdash e : \tau$

\[
\begin{align*}
(x : \tau) \in \Gamma & \quad \Rightarrow \quad \Gamma \vdash x : \tau \\
c = T | F & \quad \Rightarrow \quad \Gamma \vdash c : \text{bool} \\
\Gamma \vdash e : \text{nat} & \quad \Rightarrow \quad \Gamma \vdash \text{zero}\ e : \text{bool}
\end{align*}
\]
Typing judgement: \( \Gamma \vdash e : \tau \)

- \((x : \tau) \in \Gamma \) \( \Rightarrow \) \( \Gamma \vdash x : \tau \)
- \(c = T \mid F\) \( \Rightarrow \) \( \Gamma \vdash c : bool\)
- \(\Gamma \vdash e : nat\)
- \(\Gamma \vdash zero\)
- \(\Gamma, x : \tau \vdash e : \tau'\) \( \Rightarrow \) \(\Gamma \vdash \lambda x : \tau \rightarrow e : \tau \rightarrow \tau'\)

\(\Gamma = \{x_1 : \tau_1, \ldots, x_n : \tau_n\}\)
Typing judgement: $\Gamma \vdash e : \tau$

- $(x : \tau) \in \Gamma \quad \frac{}{\Gamma \vdash x : \tau}$
- $c = T \mid F \quad \frac{}{\Gamma \vdash c : \text{bool}}$
- $\Gamma = \{x_1 : \tau_1, \ldots, x_n : \tau_n\}$
- $\Gamma \vdash e : \text{nat}$
- $\Gamma \vdash \text{zero} e : \text{bool}$

- $\Gamma, x : \tau \vdash e : \tau'$
- $\Gamma \vdash \lambda x : \tau \rightarrow e : \tau \rightarrow \tau'$

- $\Gamma \vdash e : \tau \rightarrow \tau$
- $\Gamma \vdash \text{fix} e : \tau$
Typing judgement: \( \Gamma \vdash e : \tau \)

\[
\begin{align*}
(x : \tau) \in \Gamma & \quad \Rightarrow \quad \Gamma \vdash x : \tau \\
c = T | F & \quad \Rightarrow \quad \Gamma \vdash c : \text{bool} \\
\Gamma \vdash e : \text{nat} & \quad \Rightarrow \quad \Gamma \vdash \text{zero } e : \text{bool} \\
\Gamma, x : \tau \vdash e : \tau' & \quad \Rightarrow \quad \Gamma \vdash \lambda x : \tau \to e : \tau \to \tau' \\
\Gamma \vdash e : \tau \to \tau & \quad \Rightarrow \quad \Gamma \vdash \text{fix } e : \tau
\end{align*}
\]

Currying

\[
\lambda f : (\tau_1 \times \tau_2 \to \tau_3) \to \lambda x_1 : \tau_1 \to \lambda x_2 : \tau_2 \to f(x_1, x_2)
\]
PCF – Typing and Examples

Typing judgement: $\Gamma \vdash e : \tau$

\[
\begin{align*}
(x : \tau) &\in \Gamma \\
\hline
\hline
\Gamma &\vdash x : \tau \\
\hline
\hline
\end{align*}
\]

\[
\begin{align*}
c &\in \{T, F\} \\
\hline
\hline
\Gamma &\vdash c : \text{bool} \\
\hline
\hline
\Gamma &\vdash e : \text{nat} \\
\hline
\hline
\Gamma &\vdash \text{zero } e : \text{bool} \\
\hline
\hline
\Gamma, x : \tau &\vdash e : \tau' \\
\hline
\hline
\Gamma &\vdash \lambda x : \tau \rightarrow e : \tau \rightarrow \tau' \\
\hline
\hline
\Gamma &\vdash \text{fix } e : \tau \\
\hline
\hline
\end{align*}
\]

Currying

\[
\lambda f : (\tau_1 \times \tau_2 \rightarrow \tau_3) \rightarrow \lambda x_1 : \tau_1 \rightarrow \lambda x_2 : \tau_2 : f(x_1, x_2)
\]

Sum of numbers: $x + y$

\[
\text{fix}(\lambda(f : \text{nat} \rightarrow \text{nat} \rightarrow \text{nat}) \rightarrow \lambda x : \text{nat} \rightarrow \lambda y : \text{nat} \rightarrow \\
\text{if zero } y \text{ then } x \text{ else } S(f x (\text{pred } y)))
\]
PCF – Operational Semantics

The operational semantics is not compositional!
Evaluation judgement: $e \Downarrow c$
PCF – Operational Semantics

Evaluation judgement: \( e \downarrow c \)

Closed expression:

\[ c ::= T \mid F \mid 0 \mid S \mid (e, e) \mid \lambda x : \tau \rightarrow e \]
The operational semantics is not compositional!
Evaluation judgement: $e \Downarrow c$

$c ::= T \mid F \mid O \mid S \, c \mid (e, e) \mid \lambda x : \tau \to e$

$c = T \mid F \mid O \mid (e_1, e_2) \mid \lambda x : \tau \to e$

\[
\begin{align*}
  & c \Downarrow c \\
  & e \Downarrow 0 \\
  & \text{zero } e \Downarrow T
\end{align*}
\]
Evaluation judgement: $e \Downarrow c$

c ::= T | F | O | S c | (e, e) | \lambda x : \tau \to e

c = T | F | O | (e_1, e_2) | \lambda x : \tau \to e

\[ \frac{c \Downarrow c}{c \Downarrow c} \]

\[ \frac{e \Downarrow (e_1, e_2) \quad e_1 \Downarrow c}{\text{fst } e \Downarrow c} \]

\[ \frac{e \Downarrow 0}{\text{zero } e \Downarrow T} \]
PCF – Operational Semantics

Evaluation judgement: $e \downarrow c$

$c ::= T \mid F \mid 0 \mid S \ c \mid (e \ , \ e) \mid \lambda x : \tau \to e$

The operational semantics is not compositional!

Closed expression

$c = T \mid F \mid 0 \mid (e_1 \ , \ e_2) \mid \lambda x : \tau \to e$

$e \downarrow 0$

zero $e \downarrow T$

$e \downarrow (e_1 \ , \ e_2)$

$fst \ e \downarrow c$

$e_1 \downarrow c$

$e_1 \downarrow \lambda x : \tau \to e$

$e[e_2/x] \downarrow c$

$e_1 \ e_2 \downarrow c$
PCF – Operational Semantics

Evaluation judgement: \( e \Downarrow c \)

\[ c ::= T \mid F \mid 0 \mid S \ c \mid (e, e) \mid \lambda x : \tau \rightarrow e \]

\[
\begin{align*}
\frac{c = T \mid F \mid 0 \mid (e_1, e_2) \mid \lambda x : \tau \rightarrow e}{c \Downarrow c} & \quad \frac{e \Downarrow 0}{\text{zero } e \Downarrow T} \\
\frac{e \Downarrow (e_1, e_2)}{\text{fst } e \Downarrow c} & \quad \frac{e_1 \Downarrow \lambda x : \tau \rightarrow e}{e[e_2/x] \Downarrow c} \\
\frac{e_1 \Downarrow \lambda x : \tau \rightarrow e}{e_1 e_2 \Downarrow c} & \quad \frac{e(fix e) \Downarrow c}{\text{fix } e \Downarrow c}
\end{align*}
\]
The operational semantics is not compositional!
Domains and Continuity

Domain

- directed-complete partial order with bottom
  - directed: every pair of elements has an upper bound
  - directed-complete: every directed subset $S$ has a join ($\sqcup S$)
  - bottom: least element of the partial order, written as $\bot$

Continuous function

- directed-join-preserving function
  - directed join: join $\sqcup S$ of a directed subset $S$
  - directed-join-preserving: $f(\sqcup S) = \sqcup \{ f(d) | d \in S \}$

- point-wise order: $f \sqsubseteq g$ iff ($\forall d$) $f(d) \sqsubseteq g(d)$

- fixed point: $f(x) = x$

Steffen Lösch and Andrew M. Pitts  Full Abstraction for PCF with Names
Domains and Continuity

\[ \text{Domain} = \text{directed-complete partial order with bottom} \]
Domains and Continuity

\[ Domain = \text{directed-complete partial order with bottom} \]

- **Partial order**: set with reflexive, transitive and anti-symmetric relation \( \sqsubseteq \)
Domains and Continuity

Domain = directed-complete partial order with bottom

- **partial order**: set with reflexive, transitive and anti-symmetric relation \( \sqsubseteq \)
- **directed**: every pair of elements has an upper bound
Domains and Continuity

*Domain* = directed-complete partial order with bottom

- **partial order**: set with reflexive, transitive and anti-symmetric relation \(\sqsubseteq\)
- **directed**: every pair of elements has an upper bound
- **directed-complete**: every directed subset \(S\) has a join (\(=\)least upper bound, written as \(\bigsqcup S\))
Domains and Continuity

Domain = directed-complete partial order with bottom

- **partial order**: set with reflexive, transitive and anti-symmetric relation \( \sqsubseteq \)
- **directed**: every pair of elements has an upper bound
- **directed-complete**: every directed subset \( S \) has a join (=least upper bound, written as \( \sqcup S \))
- **bottom**: least element of the partial order, written as \( \perp \)
Domains and Continuity

*Domain* = directed-complete partial order with bottom

- **partial order**: set with reflexive, transitive and anti-symmetric relation \(\sqsubseteq\)
- **directed**: every pair of elements has an upper bound
- **directed-complete**: every directed subset \(S\) has a join (=least upper bound, written as \(\bigvee S\))
- **bottom**: least element of the partial order, written as \(\bot\)

*Continuous function* = directed-join-preserving function
Domains and Continuity

**Domain** = directed-complete partial order with bottom

- **partial order**: set with reflexive, transitive and anti-symmetric relation $\sqsubseteq$
- **directed**: every pair of elements has an upper bound
- **directed-complete**: every directed subset $S$ has a join ($=\text{least upper bound, written as } \bigsqcup S$)
- **bottom**: least element of the partial order, written as $\bot$

**Continuous function** = directed-join-preserving function

- **directed join**: join $\bigsqcup S$ of a directed subset $S$
Domains and Continuity

**Domain** = directed-complete partial order with bottom

- **partial order**: set with reflexive, transitive and anti-symmetric relation \(\sqsubseteq\)
- **directed**: every pair of elements has an upper bound
- **directed-complete**: every directed subset \(S\) has a join (≡ least upper bound, written as \(\bigvee S\))
- **bottom**: least element of the partial order, written as \(\bot\)

**Continuous function** = directed-join-preserving function

- **directed join**: join \(\bigvee S\) of a directed subset \(S\)
- **directed-join-preserving**: \(f(\bigvee S) = \bigvee\{f(d) \mid d \in S\}\)
Domains and Continuity

**Domain** = directed-complete partial order with bottom

- **partial order**: set with reflexive, transitive and anti-symmetric relation $\sqsubseteq$
- **directed**: every pair of elements has an upper bound
- **directed-complete**: every directed subset $S$ has a join (=least upper bound, written as $\bigcup S$)
- **bottom**: least element of the partial order, written as $\bot$

**Continuous function** = directed-join-preserving function

- **directed join**: join $\bigcup S$ of a directed subset $S$
- **directed-join-preserving**: $f(\bigcup S) = \bigcup\{f(d) \mid d \in S\}$
- **point-wise order**: $f \sqsubseteq g$ iff $(\forall d) f(d) \sqsubseteq g(d)$
Domains and Continuity

\textit{Domain} = directed-complete partial order with bottom

- **partial order**: set with reflexive, transitive and anti-symmetric relation $\sqsubseteq$
- **directed**: every pair of elements has an upper bound
- **directed-complete**: every directed subset $S$ has a join (=least upper bound, written as $\bigsqcup S$)
- **bottom**: least element of the partial order, written as $\bot$

Continuous function = directed-join-preserving function

- **directed join**: join $\bigsqcup S$ of a directed subset $S$
- **directed-join-preserving**: $f(\bigsqcup S) = \bigsqcup \{f(d) \mid d \in S\}$
- **point-wise order**: $f \sqsubseteq g$ iff $(\forall d) f(d) \sqsubseteq g(d)$
- **fixed point**: $f(x) = x$
Types denote domains:

\[ J \text{bool} K = \{ \text{true}, \text{false} \} \]
\[ J \text{nat} K = \{ 0, 1, 2, \ldots \} \]
\[ J \tau_1 \times \tau_2 K = J \tau_1 K \times J \tau_2 K \]

Expressions denote continuous functions:

\[ J \text{fix} e K \rho = \text{fix} (J e K \rho) \]
\[ J x K \rho = \rho x \]
\[ J T K \rho = \text{true} \]
\[ J S e K \rho = \begin{cases} n + 1 & \text{if } J e K \rho = n \\ \bot & \text{otherwise} \end{cases} \]
\[ J \lambda x : \tau . e K \rho = \lambda d \in J \tau K . J e (\rho \left[ x \mapsto d \right]) \]

domain of point-wise ordered continuous functions
Γ ⊢ e : τ ⇒ [e] ∈ [Γ] →c [τ]
\[ \Gamma \vdash e : \tau \Rightarrow [e] \in [\Gamma] \to_c [\tau] \]

domain of point-wise ordered continuous functions
Types denote domains:

\[ \Gamma \vdash e : \tau \Rightarrow [e] \in \llbracket \Gamma \rrbracket \rightarrow_c \llbracket \tau \rrbracket \]

domain of point-wise ordered continuous functions
Types denote domains:

- $[\text{bool}] = \{\text{true}, \text{false}\}_{\perp}$

Domain of point-wise ordered continuous functions
Types denote domains:

- $[[\text{bool}]] = \{\text{true}, \text{false}\}_\perp$
- $[[\text{nat}]] = \{0, 1, 2, \ldots\}_\perp$

\[ \Gamma \vdash e : \tau \Rightarrow [e] \in [\Gamma] \to_c [\tau] \]

Domain of point-wise ordered continuous functions
PCF – Denotational Semantics

\[ \Gamma \vdash e : \tau \Rightarrow [e] \in [\Gamma] \to_c [\tau] \]

Types denote domains:

- \([\text{bool}] = \{\text{true}, \text{false}\}_\perp\]
- \([\text{nat}] = \{0, 1, 2, \ldots\}_\perp\]
- \([\tau_1 \times \tau_2] = [\tau_1] \times [\tau_2]\]

Domain of point-wise ordered continuous functions

Steffen Lösch and Andrew M. Pitts
Full Abstraction for PCF with Names
PCF – Denotational Semantics

\[ \Gamma \vdash e : \tau \Rightarrow [e] \in [\Gamma] \to_c [\tau] \]

Types denote domains:
- \([\text{bool}] = \{\text{true}, \text{false}\}_\perp\)
- \([\text{nat}] = \{0, 1, 2, \ldots\}_\perp\)
- \([\tau_1 \times \tau_2] = [\tau_1] \times [\tau_2]\)
- \([\tau_1 \to \tau_2] = [\tau_1] \to_c [\tau_2]\)

Domain of point-wise ordered continuous functions
PCF – Denotational Semantics

\[ \Gamma \vdash e : \tau \Rightarrow [e] \in [\Gamma] \rightarrow_c [\tau] \]

Types denote domains:

- \([\text{bool}] = \{\text{true, false}\}_\bot\]
- \([\text{nat}] = \{0, 1, 2, \ldots\}_\bot\]
- \([\tau_1 \times \tau_2] = [\tau_1] \times [\tau_2]\]
- \([\tau_1 \rightarrow \tau_2] = [\tau_1] \rightarrow_c [\tau_2]\]
- \([\Gamma] = [\tau_1] \times \ldots \times [\tau_n]\) if \(\Gamma = \{x_1 : \tau_1, \ldots, x_n : \tau_n\}\)

Domain of point-wise ordered continuous functions
PCF – Denotational Semantics

$\Gamma \vdash e : \tau \Rightarrow [e] \in [\Gamma] \to_c [\tau]$  

Types denote domains:
- $[\text{bool}] = \{\text{true, false}\}_\perp$
- $[\text{nat}] = \{0, 1, 2, \ldots\}_\perp$
- $[\tau_1 \times \tau_2] = [\tau_1] \times [\tau_2]$
- $[\tau_1 \to \tau_2] = [\tau_1] \to_c [\tau_2]$
- $[\Gamma] = [\tau_1] \times \ldots \times [\tau_n]$ if $\Gamma = \{x_1 : \tau_1, \ldots, x_n : \tau_n\}$

Expressions denote continuous functions:  

domain of point-wise ordered continuous functions
PCF – Denotational Semantics

\[ \Gamma \vdash e : \tau \Rightarrow [e] \in [\Gamma] \rightarrow_c [\tau] \]

Types denote domains:

- \([\text{bool}] = \{\text{true}, \text{false}\}\) \(\bot\)
- \([\text{nat}] = \{0, 1, 2, \ldots\}\) \(\bot\)
- \([\tau_1 \times \tau_2] = [\tau_1] \times [\tau_2]\)
- \([\tau_1 \rightarrow \tau_2] = [\tau_1] \rightarrow_c [\tau_2]\)
- \([\Gamma] = [\tau_1] \times \ldots \times [\tau_n]\) if \(\Gamma = \{x_1 : \tau_1, \ldots, x_n : \tau_n\}\)

Expressions denote continuous functions:

\[ [x] \rho = \rho x \]
Types denote domains:

- $[\text{bool}] = \{\text{true, false}\}$
- $[\text{nat}] = \{0, 1, 2, \ldots\}$
- $[\tau_1 \times \tau_2] = [\tau_1] \times [\tau_2]$
- $[\tau_1 \rightarrow \tau_2] = [\tau_1] \rightarrow_c [\tau_2]$
- $[\Gamma] = [\tau_1] \times \ldots \times [\tau_n] \quad \text{if } \Gamma = \{x_1 : \tau_1, \ldots, x_n : \tau_n\}$

Expressions denote continuous functions:

- $[x] \rho = \rho \times$
- $[T] \rho = \text{true}$
PCF – Denotational Semantics

\[ \Gamma \vdash e : \tau \Rightarrow [e] \in [\Gamma] \rightarrow_c [\tau] \]

Types denote domains:
- \([\text{bool}] = \{\text{true}, \text{false}\}\)⊥
- \([\text{nat}] = \{0, 1, 2, \ldots\}\)⊥
- \([\tau_1 \times \tau_2] = [\tau_1] \times [\tau_2]\)
- \([\tau_1 \rightarrow \tau_2] = [\tau_1] \rightarrow_c [\tau_2]\)
- \([\Gamma] = [\tau_1] \times \ldots \times [\tau_n]\) if \(\Gamma = \{x_1 : \tau_1, \ldots, x_n : \tau_n\}\)

Expressions denote continuous functions:

\[
[x]_{\rho} = \rho x \quad [\text{T}]_{\rho} = \text{true} \quad [\text{S} \ e]_{\rho} = \begin{cases} n + 1 & \text{if } [e]_{\rho} = n \\ \bot & \text{otherwise} \end{cases}
\]
PCF – Denotational Semantics

\[ \Gamma \vdash e : \tau \Rightarrow [e] \in [\Gamma] \rightarrow_c [\tau] \]

Types denote domains:

- \([\text{bool}] = \{\text{true}, \text{false}\}_\perp\)
- \([\text{nat}] = \{0, 1, 2, \ldots\}_\perp\)
- \([\tau_1 \times \tau_2] = [\tau_1] \times [\tau_2]\)
- \([\tau_1 \rightarrow \tau_2] = [\tau_1] \rightarrow_c [\tau_2]\)
- \([\Gamma] = [\tau_1] \times \ldots \times [\tau_n] \quad \text{if} \quad \Gamma = \{x_1 : \tau_1, \ldots, x_n : \tau_n\}\)

Expressions denote continuous functions:

\[
[x]_{\rho} = \rho x \quad [T]_{\rho} = \text{true} \quad [S e]_{\rho} = \begin{cases} 
  n + 1 & \text{if} \quad [e]_{\rho} = n \\
  \perp & \text{otherwise}
\end{cases}
\]

\[
[\lambda x : \tau \rightarrow e]_{\rho} = \lambda d \in [\tau]. [e](\rho[x \mapsto d])
\]
PCF – Denotational Semantics

\[ \Gamma \vdash e : \tau \Rightarrow [e] \in [\Gamma] \rightarrow_c [\tau] \]

Types denote domains:

- \([\text{bool}] = \{\text{true}, \text{false}\}_\perp\)
- \([\text{nat}] = \{0, 1, 2, \ldots\}_\perp\)
- \([\tau_1 \times \tau_2] = [\tau_1] \times [\tau_2]\)
- \([\tau_1 \rightarrow \tau_2] = [\tau_1] \rightarrow_c [\tau_2]\)
- \([\Gamma] = [\tau_1] \times \ldots \times [\tau_n]\) if \(\Gamma = \{x_1 : \tau_1, \ldots, x_n : \tau_n\}\)

Expressions denote continuous functions:

\[
[\text{fix } e]\rho = \text{fix}([e]\rho) \quad \text{where} \quad \text{fix}(f) = \bigsqcup \{f^n(\perp) \mid n \geq 0\}
\]

\[
[x]\rho = \rho x \quad [T]\rho = \text{true} \quad [S \ e]\rho = \begin{cases} n + 1 & \text{if } [e]\rho = n \\ \bot & \text{otherwise} \end{cases}
\]

\[
[\lambda x : \tau \rightarrow e]\rho = \lambda d \in [\tau]. [e](\rho[x \mapsto d])
\]

domain of point-wise ordered continuous functions
PCF – Semantic Properties

Contextual equivalence

\[ e_1 \sim =_{\text{PCF}} e_2 \text{ iff } C[e_1] \Downarrow c \iff C[e_2] \Downarrow c \text{ for every context } C \]

We obtain a theorem and proof principle

Adequacy:

\[ J e_1 K = J e_2 K \implies e_1 \sim =_{\text{PCF}} e_2 \]

Full abstraction fails for PCF (Plotkin 1977)

\[ J e_1 K = J e_2 K \not\iff e_1 \sim =_{\text{PCF}} e_2 . \]

Observation: all PCF-domains are Scott domains

Scott domain:

- bounded-complete
- \( \omega \)-algebraic domain

- bounded-complete: bounded subsets have a join
- compact element: \( u \) is compact if for each directed \( S \) it holds that \( u \sqsubseteq \bigvee S \Rightarrow (\exists d \in S) u \sqsubseteq d \)

- \( \omega \)-algebraic: every domain element is a join of compact elements, of which there are only countably many

- \( \omega \)-algebraic: compact elements form a countable set
PCF – Semantic Properties

**Contextual equivalence**

\[ e_1 \simeq_{\text{PCF}} e_2 \iff C[e_1] \Downarrow c \iff C[e_2] \Downarrow c \text{ for every context } C[-] \]
Contextual equivalence

\[ e_1 \cong_{\text{PCF}} e_2 \iff C[e_1] \downarrow c \iff C[e_2] \downarrow c \text{ for every context } C[\_]. \]

We obtain a theorem and proof principle

**Adequacy:** \[ \llbracket e_1 \rrbracket = \llbracket e_2 \rrbracket \Rightarrow e_1 \cong_{\text{PCF}} e_2 \]
PCF – Semantic Properties

**Contextual equivalence**

\[ e_1 \cong_{\text{PCF}} e_2 \quad \text{iff} \quad C[e_1] \Downarrow c \iff C[e_2] \Downarrow c \quad \text{for every context} \quad C[\_] \]

We obtain a theorem and proof principle

**Adequacy:** \([e_1] = [e_2] \Rightarrow e_1 \cong_{\text{PCF}} e_2\)

**Full abstraction** fails for PCF (Plotkin 1977)

\([e_1] = [e_2] \neq e_1 \cong_{\text{PCF}} e_2\).
PCF – Semantic Properties

**Contextual equivalence**

\[ e_1 \simeq_{\text{PCF}} e_2 \iff C[e_1] \Downarrow c \iff C[e_2] \Downarrow c \text{ for every context } C[-] \]

We obtain a theorem and proof principle

**Adequacy:** \([e_1] = [e_2] \Rightarrow e_1 \simeq_{\text{PCF}} e_2\)

**Full abstraction** fails for PCF (Plotkin 1977)

\([e_1] = [e_2] \not\Rightarrow e_1 \simeq_{\text{PCF}} e_2\).

**Observation:** all PCF-domains are Scott domains

*Scott domain* = bounded-complete, \(\omega\)-algebraic domain
Contextual equivalence

\[ e_1 \simeq_{\text{PCF}} e_2 \iff C[e_1] \Downarrow c \iff C[e_2] \Downarrow c \text{ for every context } C[\_] \]

We obtain a theorem and proof principle

**Adequacy:** \([e_1] = [e_2] \implies e_1 \simeq_{\text{PCF}} e_2\)

**Full abstraction** fails for PCF (Plotkin 1977)

\([e_1] = [e_2] \not\implies e_1 \simeq_{\text{PCF}} e_2.\)

Observation: all PCF-domains are Scott domains

**Scott domain** = bounded-complete, \(\omega\)-algebraic domain

- **bounded-complete:** bounded subsets have a join
PCF – Semantic Properties

Contextual equivalence

\[ e_1 \simeq_{\text{PCF}} e_2 \iff C[e_1] \Downarrow c \iff C[e_2] \Downarrow c \text{ for every context } C[\cdot] \]

We obtain a theorem and proof principle

**Adequacy:** \([e_1] = [e_2] \Rightarrow e_1 \simeq_{\text{PCF}} e_2\)

**Full abstraction** fails for PCF (Plotkin 1977)

\([e_1] = [e_2] \not\Rightarrow e_1 \simeq_{\text{PCF}} e_2\]

**Observation:** all PCF-domains are Scott domains

**Scott domain** = bounded-complete, \(\omega\)-algebraic domain

- **bounded-complete:** bounded subsets have a join
- **compact element:** \(u\) is compact if for each directed \(S\) it holds that \(u \subseteq \bigsqcup S \Rightarrow (\exists d \in S) u \subseteq d\)
Contextual equivalence

\[ e_1 \cong_{\text{PCF}} e_2 \text{ iff } C[e_1] \downarrow c \iff C[e_2] \downarrow c \text{ for every context } C[-] \]

We obtain a theorem and proof principle

**Adequacy:** \[ [e_1] = [e_2] \Rightarrow e_1 \cong_{\text{PCF}} e_2 \]

**Full abstraction** fails for PCF (Plotkin 1977)

\[ [e_1] = [e_2] \not\Rightarrow e_1 \cong_{\text{PCF}} e_2. \]

Observation: all PCF-domains are Scott domains

**Scott domain** = bounded-complete, \( \omega \)-algebraic domain

- **bounded-complete**: bounded subsets have a join
- **compact element**: \( u \) is compact if for each directed \( S \) it holds that \( u \sqsubseteq \bigsqcup S \Rightarrow (\exists d \in S) \ u \sqsubseteq d \)
- **\( \omega \)-algebraic domain**: every domain element is a join of compact elements, of which there are only countably many
PCF – Semantic Properties

**Contextual equivalence**

\[ e_1 \simeq_{PCF} e_2 \text{ iff } C[e_1] \Downarrow c \iff C[e_2] \Downarrow c \text{ for every context } C[-] \]

We obtain a theorem and proof principle

**Adequacy:** \([e_1] = [e_2] \Rightarrow e_1 \simeq_{PCF} e_2\)

**Full abstraction** fails for PCF (Plotkin 1977)

\([e_1] = [e_2] \not\Rightarrow e_1 \simeq_{PCF} e_2\).

**Observation:** all PCF-domains are Scott domains

*Scott domain* = bounded-complete, \(\omega\)-algebraic domain

- **bounded-complete:** bounded subsets have a join
- **compact element:** \(u\) is compact if for each directed \(S\) it holds that \(u \sqsubseteq \bigsqcup S \Rightarrow (\exists d \in S) \, u \sqsubseteq d\)
- **\(\omega\)-algebraic domain:** every domain element is a join of compact elements, of which there are only countably many
- **\(\omega\)-algebraic:** compact elements form a countable set
PCF+ Parallel-Or

Extend PCF to PCF+ by adding ‘parallel-or’
PCF+ Parallel-Or

Extend PCF to PCF+ by adding ‘parallel-or’

\[ e ::= T \mid \ldots \mid \text{fix } e \mid e \text{ por } e \]
PCF+ Parallel-Or

Extend PCF to PCF+ by adding ‘parallel-or’

\[ e ::= T \mid \ldots \mid \text{fix } e \mid e \text{ por } e \]

\[\Gamma \vdash e_1 : \text{bool} \quad \Gamma \vdash e_2 : \text{bool}\]

\[\Gamma \vdash e_1 \text{ por } e_2 : \text{bool}\]
Extend PCF to PCF⁺ by adding ‘parallel-or’

\[
e ::= T \mid \ldots \mid \text{fix } e \mid e \text{ por } e
\]

\[
\Gamma \vdash e_1 : \text{bool} \quad \Gamma \vdash e_2 : \text{bool} \\
\Gamma \vdash e_1 \text{ por } e_2 : \text{bool}
\]

\[
\frac{e_1 \downarrow T}{e_1 \text{ por } e_2 \downarrow T} \quad \frac{e_2 \downarrow T}{e_1 \text{ por } e_2 \downarrow T} \quad \frac{e_1 \downarrow F}{e_1 \text{ por } e_2 \downarrow F} \quad \frac{e_2 \downarrow F}{e_1 \text{ por } e_2 \downarrow F}
\]

\[
J_e_1 \K_\rho = \begin{cases} 
\text{true} & \text{if } J_e_1 \K_\rho = \text{true} \text{ or } J_e_2 \K_\rho = \text{true} \\
\text{false} & \text{if } J_e_1 \K_\rho = \text{false} \text{ and } J_e_2 \K_\rho = \text{false} \\
\bot & \text{otherwise}
\end{cases}
\]

PCF⁺ is fully abstract:

\[
J_e_1 \K_\rho = J_e_2 \K_\rho \iff e_1 \sim = \text{PCF⁺ } e_2
\]

Proven by showing that all compact elements are definable

Steffen Lösch and Andrew M. Pitts
PCF+ Parallel-Or

Extend PCF to PCF+ by adding ‘parallel-or’

\[ e ::= T \mid \ldots \mid \text{fix } e \mid e \text{ por } e \]

\[ \Gamma \vdash e_1 : \text{bool} \quad \Gamma \vdash e_2 : \text{bool} \]
\[ \Gamma \vdash e_1 \text{ por } e_2 : \text{bool} \]

\[ \begin{array}{c}
e_1 \Downarrow T \\ e_1 \text{ por } e_2 \Downarrow T \\
\end{array} \quad \begin{array}{c}
e_2 \Downarrow T \\ e_1 \text{ por } e_2 \Downarrow T \\
\end{array} \quad \begin{array}{c}
e_1 \Downarrow F \\ e_1 \text{ por } e_2 \Downarrow F \\
\end{array} \quad \begin{array}{c}
e_2 \Downarrow F \\ e_1 \text{ por } e_2 \Downarrow F \\
\end{array} \]

\[ [e_1 \text{ por } e_2] \rho = \begin{cases} 
\text{true} & \text{if } [e_1] \rho = \text{true or } [e_2] \rho = \text{true} \\
\text{false} & \text{if } [e_1] \rho = \text{false and } [e_2] \rho = \text{false} \\
\bot & \text{otherwise;}
\end{cases} \]
Extend PCF to PCF+ by adding ‘parallel-or’

\[ e ::= T \mid \ldots \mid \text{fix } e \mid e \text{ por } e \]

\[
\Gamma \vdash e_1 : \text{bool} \quad \Gamma \vdash e_2 : \text{bool} \quad \Gamma \vdash e_1 \text{ por } e_2 : \text{bool}
\]

\[
\begin{align*}
& e_1 \Downarrow T \quad e_2 \Downarrow T \quad e_1 \Downarrow F \quad e_2 \Downarrow F \\
& e_1 \text{ por } e_2 \Downarrow T \\
& e_1 \text{ por } e_2 \Downarrow T \\
& e_1 \text{ por } e_2 \Downarrow F
\end{align*}
\]

\[
\llbracket e_1 \text{ por } e_2 \rrbracket \rho = \begin{cases} 
\text{true} & \text{if } \llbracket e_1 \rrbracket \rho = \text{true} \text{ or } \llbracket e_2 \rrbracket \rho = \text{true} \\
\text{false} & \text{if } \llbracket e_1 \rrbracket \rho = \text{false} \text{ and } \llbracket e_2 \rrbracket \rho = \text{false} \\
\bot & \text{otherwise;}
\end{cases}
\]

PCF+ is fully abstract: \[ \llbracket e_1 \rrbracket = \llbracket e_2 \rrbracket \iff e_1 \equiv_{\text{PCF}+} e_2 \]
PCF+ Parallel-Or

Extend PCF to PCF+ by adding ‘parallel-or’

\[ e ::= T | \ldots | \text{fix } e | e \text{ por } e \]

\[
\Gamma \vdash e_1 : \text{bool} \quad \Gamma \vdash e_2 : \text{bool}
\]

\[
\Gamma \vdash e_1 \text{ por } e_2 : \text{bool}
\]

\[
\begin{align*}
e_1 \Downarrow T & \quad \quad e_1 \text{ por } e_2 \Downarrow T \\
e_2 \Downarrow T & \quad \quad e_2 \Downarrow T \\
e_1 \Downarrow F & \quad \quad e_2 \Downarrow F
\end{align*}
\]

\[
\llbracket e_1 \text{ por } e_2 \rrbracket_\rho = \begin{cases} 
\text{true} & \text{if } \llbracket e_1 \rrbracket_\rho = \text{true} \text{ or } \llbracket e_2 \rrbracket_\rho = \text{true} \\
\text{false} & \text{if } \llbracket e_1 \rrbracket_\rho = \text{false} \text{ and } \llbracket e_2 \rrbracket_\rho = \text{false} \\
\bot & \text{otherwise;}
\end{cases}
\]

PCF+ is fully abstract: \[\llbracket e_1 \rrbracket = \llbracket e_2 \rrbracket \iff e_1 \equiv_{\text{PCF+}} e_2\]

proved by showing that all compact elements are definable
Overview

1. Introduction

2. PCF
   - language and operational semantics
   - domain theory and denotational semantics for PCF
   - parallel or and full abstraction

3. PCF with names = PNA
   - motivation and language
   - operational semantics

4. Nominal domain theory and full abstraction for PNA
PNA – Motivation

PCF can express computation over booleans, numbers. Lists or trees are not hard to add. What about expressing computation over another programming language with binding forms? This would involve having a meta-level and an object-level language. This often introduces tedious issues with $\alpha$-equivalence in object-level syntax. This is important for domain-specific languages and mechanised theorem proving.
PCF can express computation over
- booleans, numbers
- lists or trees are not hard to add
PCF can express computation over

- booleans, numbers
- lists or trees are not hard to add

What about expressing computation over another programming language with binding forms?
PCF can express computation over
- booleans, numbers
- lists or trees are not hard to add

What about expressing computation over another programming language with binding forms?
- have a meta-level and an object-level language
PCF can express computation over
- booleans, numbers
- lists or trees are not hard to add

What about expressing computation over another programming language with binding forms?
- have a meta-level and an object-level language
- often tedious issues with $\alpha$-equivalence in object-level syntax
PCF can express computation over

▶ booleans, numbers
▶ lists or trees are not hard to add

What about expressing computation over another programming language with binding forms?

▶ have a meta-level and an object-level language
▶ often tedious issues with $\alpha$-equivalence in object-level syntax
▶ important for domain-specific languages and mechanised theorem proving
Our solution: have names and name abstractions explicitly in the meta-level syntax!
Our solution: have names and name abstractions explicitly in the meta-level syntax!

PNA: Programming with Name Abstractions

Types $\tau ::= \text{bool} \mid \text{nat} \mid \tau \times \tau \mid \tau \rightarrow \tau \mid \text{name} \mid \text{term} \mid \delta \tau$

Expressions $e \in \text{Exp} ::= T \mid F \mid ... \mid \text{fix } e \mid \text{as for PCF } a \mid \nu a.e \mid \text{atomic name and local name } (e \leftrightarrow e) e \mid e = e \mid \text{name swapping and equality } V e \mid A e e \mid L e \mid \lambda\text{-terms case } e \text{ of } (V x e \mid A x x e \mid L x e) \mid \lambda\text{-term case } \alpha a.e \mid e@e \mid \text{name abstraction and concretion}$

Steffen Lösch and Andrew M. Pitts

Full Abstraction for PCF with Names
Our solution: have names and name abstractions explicitly in the meta-level syntax!

PNA: Programming with Name Abstractions

- **Types**
  \[ \tau ::= \text{bool} \mid \text{nat} \mid \tau \times \tau \mid \tau \rightarrow \tau \mid \text{name} \mid \text{term} \mid \delta \tau \]

- **Expressions**
  \[ e \in \text{Exp} ::= T \mid F \mid \ldots \mid \text{fix } e \mid \] as for PCF
PNA – Syntax

Our solution: have names and name abstractions explicitly in the meta-level syntax!

PNA: Programming with Name Abstractions

- **Types** \( \tau ::= \text{bool} \mid \text{nat} \mid \tau \times \tau \mid \tau \to \tau \mid \text{name} \mid \text{term} \mid \delta \tau \)

- **Expressions**
  
  \( e \in \text{Exp} ::= T \mid F \mid \ldots \mid \text{fix } e \mid \text{as for PCF} \)
  
  \( a \mid \nu a. \; e \mid \text{atomic name and local name} \)
Our solution: have names and name abstractions explicitly in the meta-level syntax!

PNA: Programming with Name Abstractions

- **Types** \( \tau ::= \text{bool} \mid \text{nat} \mid \tau \times \tau \mid \tau \rightarrow \tau \mid \text{name} \mid \text{term} \mid \delta \tau \)**

- **Expressions**

  \[ e \in \text{Exp} ::= T \mid F \mid \ldots \mid \text{fix } e \mid \text{as for PCF} \]
  \[ a \mid \nu a. \ e \mid \text{atomic name and local name} \]
  \[ (e \leftrightarrow e) \ e \mid e = e \mid \text{name swapping and equality} \]
Our solution: have names and name abstractions explicitly in the meta-level syntax!

PNA: Programming with Name Abstractions

- **Types** \( \tau ::= \text{bool} \mid \text{nat} \mid \tau \times \tau \mid \tau \rightarrow \tau \mid \text{name} \mid \text{term} \mid \delta \tau \)

- **Expressions**
  
  \( e \in \text{Exp} ::= T \mid F \mid \ldots \mid \text{fix } e \mid \)  
  \( a \mid \nu a. \ e \mid (e \equiv e) \ e \mid e = e \mid V \ e \mid A \ e \ e \mid L \ e \mid \)

  *as for PCF*

  atomic name and local name
  
  name swapping and equality
  
  \( \lambda \)-terms
Our solution: have names and name abstractions explicitly in the meta-level syntax!

PNA: Programming with Name Abstractions

- **Types**
  \( \tau ::= \text{bool} \mid \text{nat} \mid \tau \times \tau \mid \tau \to \tau \mid \text{name} \mid \text{term} \mid \delta \tau \)

- **Expressions**

  \( e \in \text{Exp} ::= T \mid F \mid \ldots \mid \text{fix } e \mid a \mid \nu a. e \mid (e \leftrightarrow e) \mid e = e \mid \text{atomic name and local name} \)

  \( \text{V } e \mid \text{A } e e \mid \text{L } e \mid \text{name swapping and equality} \)

  \( \text{case } e \text{ of } (\text{V } x \to e \mid \text{A } x x \to e \mid \text{L } x \to e) \mid \lambda \text{-terms} \)

  \( \text{case } e \text{ of } (\text{V } x \to e \mid \text{A } x x \to e \mid \text{L } x \to e) \mid \lambda \text{-term case} \)

Steffen Lösch and Andrew M. Pitts  
Full Abstraction for PCF with Names
Our solution: have names and name abstractions explicitly in the meta-level syntax!

PNA: Programming with Name Abstractions

- **Types** \( \tau ::= \text{bool} \mid \text{nat} \mid \tau \times \tau \mid \tau \rightarrow \tau \mid \text{name} \mid \text{term} \mid \delta \tau \)

- **Expressions**
  
  \( e \in \text{Exp} ::= T \mid F \mid \ldots \mid \text{fix } e \mid \text{as for PCF} \)

  \( a \mid \nu a. \ e \mid \text{atomic name and local name} \)

  \( (e \equiv e) \ e \mid e = e \mid \text{name swapping and equality} \)

  \( V \ e \mid A \ e \ e \mid L \ e \mid \text{\( \lambda \)-terms} \)

  \( \text{case } e \text{ of } (V x \rightarrow e \mid A x x \rightarrow e \mid L x \rightarrow e) \mid \text{\( \lambda \)-term case} \)

  \( \alpha a. \ e \mid e @ e \mid \text{name abstraction and concretion} \)
PNA – Syntax

Our solution: have names and name abstractions explicitly in the meta-level syntax!

PNA: Programming with Name Abstractions

- **Types** \( \tau ::= \text{bool} | \text{nat} | \tau \times \tau | \tau \rightarrow \tau | \text{name} | \text{term} | \delta \tau \)

- **Expressions**

  \( e \in \text{Exp} ::= T | F | \ldots | \text{fix } e | \text{as for PCF} \)

  \( a \mid \nu a. e \mid (e \rightleftharpoons e) \mid e = e \mid \text{atomic name and local name} \)

  \( V e \mid A e e \mid L e \mid \lambda\text{-terms} \)

  case \( e \) of (\( V x \rightarrow e \mid A x x \rightarrow e \mid L x \rightarrow e \)) \mid \text{\( \lambda\)-term case} \)

  \( \alpha a. e \mid e @ e \mid \text{name abstraction and concretion} \)

- **Binding forms** \( \lambda x : \tau \rightarrow _\_ , \nu a. _\_ , \alpha a. _\_ , \)

  case \( e \) of (\( V x \rightarrow _\_ \mid A x x \rightarrow _\_ \mid L x \rightarrow _\_)
We have two kinds of identifiers now

- Variables 
- Names

\( x \in V \) 
\( a \in A \)

- can be substituted
- can be permuted

\( A \cap V = \emptyset \)

\( V \) and \( A \) are countably infinite

- both identifiers can be bound

Perm \( A \)-action (finite name permutation) on syntax:

\[
\begin{align*}
\pi \cdot x &= x \\
\pi \cdot a &= \pi(a) \\
\pi \cdot T &= T \\
\pi \cdot (e_1, e_2) &= (\pi \cdot e_1, \pi \cdot e_2) \\
\pi \cdot \nu a.e &= \nu(\pi \cdot a).\pi \cdot e
\end{align*}
\]

Steffen L"osch and Andrew M. Pitts

Full Abstraction for PCF with Names
We have two kinds of identifiers now

<table>
<thead>
<tr>
<th>Variables</th>
<th>Names</th>
</tr>
</thead>
<tbody>
<tr>
<td>$x \in \mathbb{V}$</td>
<td>$a \in \mathbb{A}$</td>
</tr>
<tr>
<td>can be substituted</td>
<td>can be permuted</td>
</tr>
<tr>
<td>$\mathbb{A} \cap \mathbb{V} = \emptyset$</td>
<td></td>
</tr>
<tr>
<td>$\mathbb{V}$ and $\mathbb{A}$ are countably infinite</td>
<td></td>
</tr>
<tr>
<td>both identifiers can be bound</td>
<td></td>
</tr>
</tbody>
</table>
PNA – Names and Variables

We have two kinds of identifiers now

<table>
<thead>
<tr>
<th>Variables</th>
<th>Names</th>
</tr>
</thead>
<tbody>
<tr>
<td>$x \in \mathcal{V}$</td>
<td>$a \in \mathcal{A}$</td>
</tr>
<tr>
<td>can be substituted</td>
<td>can be permuted</td>
</tr>
<tr>
<td>$\mathcal{A} \cap \mathcal{V} = \emptyset$</td>
<td></td>
</tr>
<tr>
<td>$\mathcal{V}$ and $\mathcal{A}$ are countably infinite</td>
<td></td>
</tr>
<tr>
<td>both identifiers can be bound</td>
<td></td>
</tr>
</tbody>
</table>

**Perm$_{\mathcal{A}}$-action (finite name permutation) on syntax:**

\[
\begin{align*}
\pi \cdot x &= x \\
\pi \cdot a &= \pi(a) \\
\pi \cdot T &= T \\
\pi \cdot (e_1, e_2) &= (\pi \cdot e_1, \pi \cdot e_2) \\
\pi \cdot \nu a. e &= \nu(\pi \cdot a) \cdot \pi \cdot e
\end{align*}
\]
PNA – Typing
PNA – Typing

\[
\frac{a \in A}{\Gamma \vdash a : \text{name}}
\]
PNA – Typing

\[ a \in A \]
\[ \Gamma \vdash a : \text{name} \]

\[ \Gamma \vdash e_1 : \text{name} \quad \Gamma \vdash e_2 : \text{name} \]
\[ \Gamma \vdash e_1 = e_2 : \text{bool} \]
Steffen Lösch and Andrew M. Pitts  

Full Abstraction for PCF with Names
PNA – Typing

\[
\frac{a \in A}{\Gamma \vdash a : \text{name}} \quad \frac{a \in A \quad \Gamma \vdash e : \tau}{\Gamma \vdash \nu a. e : \tau}
\]

\[
\frac{\Gamma \vdash e_1 : \text{name} \quad \Gamma \vdash e_2 : \text{name} \quad \Gamma \vdash e_3 : \tau}{\Gamma \vdash (e_1 \Rightarrow e_2) e_3 : \tau}
\]

\[
\frac{\Gamma \vdash e_1 : \text{name} \quad \Gamma \vdash e_2 : \text{name}}{\Gamma \vdash e_1 = e_2 : \text{bool}}
\]

\[
\frac{a \in A \quad \Gamma \vdash e : \tau}{\Gamma \vdash \alpha a. e : \delta \tau}
\]

\[
\frac{\Gamma \vdash e_1 : \delta \tau \quad \Gamma \vdash e_2 : \text{name}}{\Gamma \vdash e_1 @ e_2 : \tau}
\]
PNA – Typing

\[
\begin{align*}
  & a \in A \\
  & \frac{}{\Gamma \vdash a : \text{name}} \\
  & a \in A \\
  & \frac{\Gamma \vdash e : \tau}{\Gamma \vdash \nu a. e : \tau}
\end{align*}
\]

\[
\begin{align*}
  & \Gamma \vdash e_1 : \text{name} \\
  & \frac{\Gamma \vdash e_2 : \text{name}}{\Gamma \vdash e_3 : \tau} \\
  & \frac{\Gamma \vdash (e_1 \Rightarrow e_2) e_3 : \tau}{\Gamma \vdash e_1 = e_2 : \text{bool}} \\
  & \Gamma \vdash e : \text{name} \\
  & \frac{\Gamma \vdash e_1 : \text{term} \quad \Gamma \vdash e_2 : \text{term}}{\Gamma \vdash \nu e_1 e_2 : \text{term}} \\
  & \frac{\Gamma \vdash e : \delta \text{term}}{\Gamma \vdash V e : \text{term}} \\
  & \frac{\Gamma \vdash e : \text{name} \quad \Gamma \vdash e_1 : \text{term}}{\Gamma \vdash A e_1 e_2 : \text{term}} \\
  & \frac{\Gamma \vdash e : \text{term}}{\Gamma \vdash L e : \text{term}} \\
\end{align*}
\]

\[
\begin{align*}
  & a \in A \\
  & \frac{\Gamma \vdash e : \tau}{\Gamma \vdash \alpha a. e : \delta \tau} \\
  & \Gamma \vdash e_1 : \delta \tau \\
  & \frac{\Gamma \vdash e_2 : \text{name}}{\Gamma \vdash e_1 \odot e_2 : \tau}
\end{align*}
\]

Steffen Lösch and Andrew M. Pitts  
Full Abstraction for PCF with Names
PNA – Typing

\[
\begin{align*}
\frac{a \in A}{\Gamma \vdash a : \text{name}} & \quad \frac{a \in A \quad \Gamma \vdash e : \tau}{\Gamma \vdash \nu a. e : \tau} \\
\frac{\Gamma \vdash e_1 : \text{name} \quad \Gamma \vdash e_2 : \text{name}}{\Gamma \vdash e_3 : \tau} & \quad \frac{\Gamma \vdash e_1 : \text{name} \quad \Gamma \vdash e_2 : \text{name}}{\Gamma \vdash e_1 = e_2 : \text{bool}} \\
\frac{\Gamma \vdash e : \text{name}}{\Gamma \vdash \text{V} e : \text{term}} \quad \frac{\Gamma \vdash e_1 : \text{term} \quad \Gamma \vdash e_2 : \text{term}}{\Gamma \vdash \text{A} e_1 e_2 : \text{term}} \quad \frac{\Gamma \vdash e : \delta \text{term}}{\Gamma \vdash \text{L} e : \text{term}} \\
\frac{\Gamma \vdash e : \text{term} \quad \Gamma, x_1 : \text{name} \vdash e_1 : \tau}{\Gamma, x_2 : \text{term}, x'_2 : \text{term} \vdash e_2 : \tau} \quad \frac{\Gamma, x_3 : \delta \text{term} \vdash e_3 : \tau}{\Gamma \vdash \text{case} e \text{ of (V} x_1 \to e_1 \mid \text{A} x_2 x'_2 \to e_2 \mid \text{L} x_3 \to e_3) : \tau} \\
\frac{a \in A \quad \Gamma \vdash e : \tau}{\Gamma \vdash \alpha a. e : \delta \tau} \quad \frac{\Gamma \vdash e_1 : \delta \tau \quad \Gamma \vdash e_2 : \text{name}}{\Gamma \vdash e_1 \odot e_2 : \tau}
\end{align*}
\]
PNA – Example Programs

Object-level representation of ($\lambda a \quad a \quad b \quad c$) $A (L (\alpha a . A (V a)) (V b))$ (V c)

Capture-avoiding substitution $y' [y/x]$

$\lambda y$: term $\lambda x$: name

fix ($\lambda (f$: term $\quad term$ $\lambda y'$: term $\quad term)

case $y'$ of $V x_1$ if $x_1 = x$ then $y$ else $y'$

<table>
<thead>
<tr>
<th>A $y_2$ $y'_2$</th>
</tr>
</thead>
<tbody>
<tr>
<td>A ($f y_2$) ($f y'_2$)</td>
</tr>
</tbody>
</table>

| L $z$ $L (\alpha a . f (z \theta a))$ |

Steffen Lösch and Andrew M. Pitts  Full Abstraction for PCF with Names
PNA – Example Programs

Object-level representation of \((\lambda a \to a \ b) \ c\)

\[ A \ (L \ (\alpha . \ A \ (V \ a) \ (V \ b))) \ (V \ c) \]
Object-level representation of \((\lambda a \to a \ b) \ c\)

\[ A \ (L \ (\alpha a. \ A \ (V \ a) \ (V \ b))) \ (V \ c) \]

Capture-avoiding substitution \(y'[y/x]\)

\[ \lambda y : \text{term} \to \lambda x : \text{name} \to \]
\[ \text{fix}(\lambda(f : \text{term} \to \text{term}) \to \lambda y' : \text{term} \to \]
\[ \text{case } y' \text{ of} \]
\[ V \ x_1 \to \text{if } x_1 = x \text{ then } y \text{ else } y' \]
\[ | \ A \ y_2 \ y_2' \to A \ (f \ y_2) \ (f \ y_2') \]
\[ | \ L \ z \to L \ (\alpha a. \ f(z @ a)). \]
PNA – Operational Semantics

\[ e \Downarrow c \]
\[ a \Downarrow a \]
\[ e \Downarrow c a \]
\[ c := c' \]
\[ \nu a \cdot e \Downarrow c' \]
\[ e_1 \Downarrow e_2 \Downarrow e_3 \Downarrow c (e_1 \leftrightarrow e_2) e_3 \Downarrow (a_1 a_2) \cdot c \]
\[ \alpha a \cdot e \Downarrow \alpha a \cdot c \]
\[ e_1 \Downarrow \alpha a \cdot c e_2 \Downarrow a' \]
\[ a \neq a' \]
\[ \nu a \cdot (a a') \cdot c \Downarrow c' e_1 @ e_2 \Downarrow c' e_1 \Downarrow a e_2 \Downarrow a' \]
\[ e_1 \Downarrow T e_2 \Downarrow (a a') \cdot c \Downarrow c' e_1 \Downarrow V c e_2 \Downarrow V c \]
\[ e_1 \Downarrow c_1 e_2 \Downarrow A c_1 c_2 \]
\[ e_1 e_2 \Downarrow A c_1 c_2 \]
\[ e_1 \Downarrow c L \]
\[ e_1 \Downarrow L c \]
\[ e_1 \Downarrow \nu c \cdot \]
\[ e_1 \Downarrow c' \]
\[ \text{case } e \text{ of } (V x_1 e_1 | \cdots) \Downarrow c' \]

variable-closed expression

\[ c ::= \ldots | a | V c | A c \]

Steffen Lösch and Andrew M. Pitts
Full Abstraction for PCF with Names
Evaluation judgement: $e \Downarrow c$
Evaluation judgement: $e \Downarrow c$

$c ::= \ldots | a | Vc | Ac c | Lc | \alpha a. c$
PNA – Operational Semantics

Evaluation judgement: \( e \Downarrow c \)

\[ a \in A \]

\[ a \Downarrow a \]

variable-closed expression

\[ c ::= \ldots | a | Vc | Acc | Lc | \alpha a. c \]
Evaluation judgement: $e \Downarrow c$

$c ::= \ldots | a | V c | A c c | L c | \alpha a. c$

\[
\begin{align*}
  a &\in A \\
  \frac{}{a \Downarrow a}
\end{align*}
\]

\[
\begin{align*}
  e_1 \Downarrow a &\quad e_2 \Downarrow a \\
  \frac{}{e_1 = e_2 \Downarrow T}
\end{align*}
\]

\[
\begin{align*}
  e_1 \Downarrow a &\quad e_2 \Downarrow a' & a \neq a' \\
  \frac{}{e_1 = e_2 \Downarrow F}
\end{align*}
\]
PNA – Operational Semantics

Evaluation judgement: $e \Downarrow c$

$c ::= \ldots \mid a \mid V \ c \mid A \ c \ c \mid L \ c \mid \alpha a. \ c$

$a \in A$

$\frac{}{a \Downarrow a}$

$e_1 \Downarrow a_1 \quad e_2 \Downarrow a_2 \quad e_3 \Downarrow c$

$\frac{(e_1 = e_2) \ e_3 \Downarrow (a_1 a_2) \cdot c}{(e_1 \Downarrow a_1 \quad e_2 \Downarrow a_2 \quad e_3 \Downarrow c}$

$e_1 \Downarrow a \quad e_2 \Downarrow a'$

$\frac{a \neq a'}{e_1 = e_2 \Downarrow F}$
PNA – Operational Semantics

Evaluation judgement: $e \Downarrow c$

$c ::= \ldots \mid a \mid Vc \mid Ac \mid Lc \mid \alpha a. c$

\[
\frac{a \in A}{a \Downarrow a}
\]

\[
\frac{e_1 \Downarrow a \quad e_2 \Downarrow a}{e_1 = e_2 \Downarrow T}
\]

\[
\frac{e_1 \Downarrow c_1 \quad e_2 \Downarrow c_2}{A \ e_1 \ e_2 \Downarrow A \ c_1 \ c_2}
\]

\[
\frac{e \Downarrow c}{L \ e \Downarrow L \ c}
\]

\[
\frac{e_1 \Downarrow a_1 \quad e_2 \Downarrow a_2 \quad e_3 \Downarrow c}{(e_1 \rightleftharpoons e_2) \ e_3 \Downarrow (a_1 \ a_2) \cdot c}
\]

\[
\frac{e_1 \Downarrow a \quad e_2 \Downarrow a'}{e_1 = e_2 \Downarrow F}
\]

\[
\frac{e_1 \Downarrow a \quad e_2 \Downarrow a' \quad a \neq a'}{e \Downarrow c \quad V \ e \Downarrow V \ c}
\]
PNA – Operational Semantics

Evaluation judgement:  \( e \Downarrow c \)

\[
\begin{align*}
    a \in A & \quad \quad & \quad e_1 \Downarrow a_1, \quad e_2 \Downarrow a_2, \quad e_3 \Downarrow c \\
    a \Downarrow a & \quad \quad & \quad (e_1 = e_2) \quad e_3 \Downarrow (a_1 a_2) \cdot c \\
    e_1 \Downarrow a, \quad e_2 \Downarrow a & \quad \quad & \quad e_1 = e_2 \Downarrow T \\
    e_1 \Downarrow c_1, \quad e_2 \Downarrow c_2 & \quad \quad & \quad A e_1 e_2 \Downarrow A c_1 c_2 \\
    e \Downarrow c & \quad \quad & \quad L e \Downarrow L c \\
    e \Downarrow V c, \quad e_1[c/x_1] \Downarrow c' & \quad \quad & \quad \text{case } e \text{ of } (V x_1 \to e_1 | \cdots) \Downarrow c'
\end{align*}
\]

\[c ::= \ldots | a | V c | A c c | L c | \alpha a. c\]
PNA – Operational Semantics

Evaluation judgement: \( e \downarrow c \)

\[
\begin{align*}
\text{variable-closed expression} \\
\text{c ::= \ldots | a | V c | A c c | L | \alpha a. c}
\end{align*}
\]

\[
\begin{align*}
& a \in A \\
\frac{}{a \downarrow a}
\end{align*}
\]

\[
\begin{align*}
& e \downarrow c \\
\frac{}{\alpha a. e \downarrow \alpha a. c}
\end{align*}
\]

\[
\begin{align*}
& e_1 \downarrow a \quad e_2 \downarrow a \\
\frac{}{e_1 = e_2 \downarrow T}
\end{align*}
\]

\[
\begin{align*}
& e_1 \downarrow a \quad e_2 \downarrow a' \\
\frac{}{e_1 = e_2 \downarrow F}
\end{align*}
\]

\[
\begin{align*}
& e_1 \downarrow c_1 \quad e_2 \downarrow c_2 \\
\frac{}{A e_1 e_2 \downarrow A c_1 c_2}
\end{align*}
\]

\[
\begin{align*}
& e \downarrow c \\
\frac{}{L e \downarrow L c}
\end{align*}
\]

\[
\begin{align*}
& e \downarrow V c \\
\frac{}{\text{case } e \text{ of (V } x_1 \rightarrow e_1 | \cdots ) \downarrow c'}
\end{align*}
\]

\[
\begin{align*}
& e_1 \downarrow a_1 \quad e_2 \downarrow a_2 \quad e_3 \downarrow c \\
\frac{}{(e_1 \rightleftharpoons e_2) e_3 \downarrow (a_1 a_2) \cdot c}
\end{align*}
\]

e \downarrow V c \\
\frac{}{e_1[c/x_1] \downarrow c'}
PNA – Operational Semantics

Evaluation judgement: $e \downarrow c$

$c ::= \ldots \mid a \mid Vc \mid Ac \mid Lc \mid \alpha a. c$

$$
a \in A \quad \frac{}{a \downarrow a}
$$

$$
e \downarrow c \quad \frac{e_1 \downarrow \alpha a. c \quad e_2 \downarrow a' \quad a \neq a'}{\nu a. (a a') \cdot c \downarrow c'}
$$

$$
e_1 \downarrow a \quad e_2 \downarrow a \quad \frac{}{e_1 = e_2 \downarrow T}
$$

$$
e_1 \downarrow a \quad e_2 \downarrow a' \quad a \neq a' \quad \frac{}{e_1 = e_2 \downarrow F}
$$

$$
e_1 \downarrow c_1 \quad e_2 \downarrow c_2 \quad \frac{}{A e_1 e_2 \downarrow Ac_1 c_2}
$$

$$
e \downarrow c \quad \frac{}{L e \downarrow Lc}
$$

$$
e \downarrow Vc \quad e_1[c/x_1] \downarrow c' \quad \frac{\text{case } e \text{ of } (V x_1 \rightarrow e_1 \mid \cdots) \downarrow c'}{c'}
$$

Steffen Lösch and Andrew M. Pitts

Full Abstraction for PCF with Names
Evaluation judgement: $e \Downarrow c$

\[
\begin{align*}
\text{variable-closed expression} \\
\text{c ::= ... | a | V c | A \ c \ c | L \ c | \alpha . a . c}
\end{align*}
\]
‘Odersky-style’ partial name restriction operation $a \setminus c := c'$:
‘Odersky-style’ partial name restriction operation \( a \setminus c := c' \):

\[
c = T \mid F \mid 0 \mid S \ c'
\]

\[
a \setminus c := c
\]
‘Odersky-style’ partial name restriction operation $a \backslash c := c'$:

$$
\begin{align*}
\text{c} &= T \mid F \mid 0 \mid S \text{c'} \\
\hline
\text{a} \backslash \text{c} &:= \text{c}
\end{align*}
$$

$$
\begin{align*}
\text{a} \neq \text{a}' \\
\hline
\text{a} \backslash \text{a}' &:= \text{a}'
\end{align*}
$$
PNA – Name Restriction Operation

‘Odersky-style’ partial name restriction operation $a \setminus c := c'$:

\[
\begin{align*}
  c &= T \mid F \mid 0 \mid S \; c' \\
  a \setminus c &:= c \\
  a \setminus (e_1, e_2) &:= (\nu a. e_1, \nu a. e_2) \\
  a \setminus \lambda x : \tau \to e &:= \lambda x : \tau \to \nu a. e \\
  a \setminus a' &:= a'
\end{align*}
\]
‘Odersky-style’ partial name restriction operation $a\backslash c := c'$:

$$
c = T \mid F \mid O \mid S c' \quad \frac{a\backslash c := c}{a\backslash (e_1, e_2) := (\nu a. e_1, \nu a. e_2)}
$$

$$
a\backslash \lambda x : \tau \rightarrow e := \lambda x : \tau \rightarrow \nu a. e
$$

$$
a \neq a' \quad a\backslash a' := a' \quad a\backslash V c := V c'
$$

$$
a\backslash c_1 := c'_1 \quad a\backslash c_2 := c'_2 \quad a\backslash A c_1 c_2 := A c'_1 c'_2
$$

$$
a\backslash L c := L c'
$$
PNA – Name Restriction Operation

‘Odersky-style’ partial name restriction operation $a \backslash c := c'$:

\[
\begin{align*}
  c &= T \mid F \mid 0 \mid S \ c' \\
  a \backslash c &= c \\
  a \backslash (e_1, e_2) &= (\nu a. \ e_1, \nu a. \ e_2) \\
  a \backslash (\lambda x : \tau \to e) &= \lambda x : \tau \to \nu a. \ e \\
  a \backslash a' &= a' \\
  a \backslash V c &= V c' \\
  a \backslash c_1 &= c_1' \\
  a \backslash c_2 &= c_2' \\
  a \backslash A \ c_1 \ c_2 &= A \ c_1' \ c_2' \\
  a \backslash L \ c &= L \ c' \\
  a \backslash \alpha a'. \ c &= \alpha a'. \ c' \\
\end{align*}
\]
PNA – Name Restriction Operation

‘Odersky-style’ partial name restriction operation $a \setminus c := c'$:

$$
\begin{align*}
  c &= T \mid F \mid 0 \mid S c' \\
  a \setminus c &= c \\
  a \setminus (e_1, e_2) &= (\nu a. e_1, \nu a. e_2)
\end{align*}
$$

$$
\begin{align*}
  a \setminus \lambda x : \tau \to e &= \lambda x : \tau \to \nu a. e \\
  a \setminus a' &= a' \\
  a \setminus V c &= V c'
\end{align*}
$$

$$
\begin{align*}
  a \setminus c_1 &= c_1' \\
  a \setminus c_2 &= c_2' \\
  a \setminus A c_1 c_2 &= A c_1' c_2' \\
  a \setminus L c &= L c'
\end{align*}
$$

$$
\begin{align*}
  a \setminus c &= c' \\
  a \neq a' \\
  a \setminus \alpha a'. c &= \alpha a'. c'
\end{align*}
$$

Note: $(\not\exists c) a \setminus a = c$
Overview

1. Introduction

2. PCF
   - language and operational semantics
   - domain theory and denotational semantics for PCF
   - parallel or and full abstraction

3. PCF with names = PNA
   - motivation and language
   - operational semantics

4. Nominal domain theory and full abstraction for PNA
Nominal Partial Orders

Theme: Remodel the domain theory with nominal sets.

▶ nominal set: set with $\text{Perm}_A$-action whose elements are finitely supported; Examples: PNA-syntax, $A$ with finite support: $x$ is finitely supported by $A \subseteq f_A$ if all permutations that preserve $A$ also preserve $x$.

▶ nominal partial order: nominal set with a partial order $\sqsubseteq$ satisfying $d \sqsubseteq d' \Rightarrow \pi \cdot d \sqsubseteq \pi \cdot d'$.

Crucial construction: name abstraction sets

▶ For a nominal set $X$, define a preorder on $A \times X$ by $(a_1, x_1) \sqsubseteq (a_2, x_2)$ iff $(a_1 b) \cdot x_1 \sqsubseteq (a_2 b) \cdot x_2$ for some fresh $b$. Let $\approx$ be the according equivalence relation.

$[A]_X = (A \times X) / \approx$ is the name abstraction set of $X$ and write $\langle a \rangle x$ for the equivalence class of $(a, x)$.

$[A]_X$ is a nominal partial order with $\sqsubseteq$ as above.
Nominal Partial Orders

Theme: Remodel the domain theory with *nominal sets*.

**Nominal Set**
- A set with a $\text{Perm}_A$-action, whose elements are finitely supported.
- Examples: PNA-syntax, $A$ with finite support $x$ is finitely supported by $A \subseteq f_A$ if all permutations that preserve $A$ also preserve $x$.

**Nominal Partial Order**
- A nominal set with a partial order $\sqsubseteq$ satisfying $d \sqsubseteq d' \Rightarrow \pi \cdot d \sqsubseteq \pi \cdot d'$.

**Crucial Construction: Name Abstraction Sets**
- For a nominal set $X$, define a preorder on $A \times X$ by $(a_1, x_1) \sqsubseteq (a_2, x_2)$ iff $(a_1 b) \cdot x_1 \sqsubseteq (a_2 b) \cdot x_2$ for some fresh $b$.
- Let $\approx$ be the according equivalence relation.
- $[A] \downarrow X = (A \times X) / \approx$ is the name abstraction set of $X$, and write $\langle a \rangle x$ for the equivalence class of $(a, x)$.
- $[A] \downarrow X$ is a nominal partial order with $\sqsubseteq$ as above.
Nominal Partial Orders

Theme: Remodel the domain theory with nominal sets.

- **nominal set**: set with $\text{Perm}_A$-action $\cdot$ whose elements are finitely supported; *Examples*: PNA-syntax, $A$

Nominal Partial Orders

Theme: Remodel the domain theory with nominal sets.

- **nominal set**: set with $\text{Perm}_A$-action $\_ \cdot \_$ whose elements are finitely supported; *Examples*: PNA-syntax, $A$

- **finite support**: $x$ is finitely supported by $A \subseteq_f A$ if all permutations that preserve $A$ also preserve $x$
Nominal Partial Orders

Theme: Remodel the domain theory with nominal sets.

- **nominal set**: set with $\text{Perm}_A$-action $\cdot$ whose elements are finitely supported; *Examples*: PNA-syntax, $\mathbb{A}$

- **finite support**: $x$ is finitely supported by $A \subseteq_f \mathbb{A}$ if all permutations that preserve $A$ also preserve $x$

- **nominal partial order**: nominal set with a partial order $\sqsubseteq$ satisfying $d \sqsubseteq d' \Rightarrow \pi \cdot d \sqsubseteq \pi \cdot d'$

Steffen Lösch and Andrew M. Pitts | Full Abstraction for PCF with Names
Nominal Partial Orders

Theme: Remodel the domain theory with *nominal sets*.

- **nominal set**: set with $\text{Perm}_A$-action $\_ \cdot \_$ whose elements are finitely supported; *Examples*: PNA-syntax, $A$

- **finite support**: $x$ is finitely supported by $A \subseteq_f A$ if all permutations that preserve $A$ also preserve $x$

- **nominal partial order**: nominal set with a partial order $\sqsubseteq$ satisfying $d \sqsubseteq d' \Rightarrow \pi \cdot d \sqsubseteq \pi \cdot d'$

Crucial construction: *name abstraction sets*
Nominal Partial Orders

Theme: Remodel the domain theory with *nominal sets*. 

- **nominal set**: set with $\text{Perm}_A$-action $\_ \cdot \_$ whose elements are finitely supported; *Examples*: PNA-syntax, $\mathbb{A}$

- **finite support**: $x$ is finitely supported by $A \subseteq_f \mathbb{A}$ if all permutations that preserve $A$ also preserve $x$

- **nominal partial order**: nominal set with a partial order $\sqsubseteq$ satisfying $d \sqsubseteq d' \Rightarrow \pi \cdot d \sqsubseteq \pi \cdot d'$

Crucial construction: *name abstraction sets*

- For a nominal set $X$, define a preorder on $\mathbb{A} \times X$ by $(a_1, x_1) \sqsubseteq (a_2, x_2)$ iff $(a_1 b) \cdot x_1 \sqsubseteq (a_2 b) \cdot x_2$ for some fresh $b$. Let $\approx$ be the according equivalence relation.
Nominal Partial Orders

Theme: Remodel the domain theory with *nominal sets*.

- **nominal set**: set with $\text{Perm}_A$-action $\_ \cdot \_$ whose elements are finitely supported; *Examples*: PNA-syntax, $A$

- **finite support**: $x$ is finitely supported by $A \subseteq_f A$ if all permutations that preserve $A$ also preserve $x$

- **nominal partial order**: nominal set with a partial order $\sqsubseteq$ satisfying $d \sqsubseteq d' \Rightarrow \pi \cdot d \sqsubseteq \pi \cdot d'$

Crucial construction: *name abstraction sets*

- For a nominal set $X$, define a preorder on $A \times X$ by $(a_1, x_1) \sqsubseteq (a_2, x_2)$ iff $(a_1 \ b) \cdot x_1 \sqsubseteq (a_2 \ b) \cdot x_2$ for some fresh $b$. Let $\approx$ be the according equivalence relation.

- $[A]X = (A \times X)/\approx$ is the *name abstraction set* of $X$ and write $\langle a \rangle x$ for the equivalence class of $(a, x)$. 
Nominal Partial Orders

Theme: Remodel the domain theory with nominal sets.

- **nominal set**: set with \( \text{Perm}_A \)-action \( _{-} \cdot _{-} \) whose elements are finitely supported; *Examples*: PNA-syntax, \( A \)

- **finite support**: \( x \) is finitely supported by \( A \subseteq_f A \) if all permutations that preserve \( A \) also preserve \( x \)

- **nominal partial order**: nominal set with a partial order \( \sqsubseteq \) satisfying \( d \sqsubseteq d' \Rightarrow \pi \cdot d \sqsubseteq \pi \cdot d' \)

Crucial construction: *name abstraction sets*

- For a nominal set \( X \), define a preorder on \( A \times X \) by \( (a_1, x_1) \sqsubseteq (a_2, x_2) \) iff \( (a_1 \ b) \cdot x_1 \sqsubseteq (a_2 \ b) \cdot x_2 \) for some fresh \( b \). Let \( \approx \) be the according equivalence relation.

- \([A]X = (A \times X)/\sim \) is the **name abstraction set** of \( X \) and write \( \langle a \rangle x \) for the equivalence class of \( (a, x) \).

- \([A]X \) is a nominal partial order with \( \sqsubseteq \) as above.
Observations:
1. $\langle a \rangle \bigoplus S = \bigoplus \{\langle a \rangle d \mid d \in S\}$ does not hold for all directed (and finitely supported) sets.
2. It does hold for all uniform-directed sets (directed set where all elements are supported by a common finite set).
3. A nominal partial order is finitely-supported-chain complete iff it is uniform-directed complete.

Nominal Scott domain $\mathbb{D}^{\text{uc}, \text{fs}}$: finitely supported, bounded subsets have a join. Uniform-continuous function: uniform-directed-join preserving function; the finitely supported ones form the nominal Scott domain $\mathbb{D}^{\text{uc}, \text{fs}}$. 

Steffen Lösch and Andrew M. Pitts
Full Abstraction for PCF with Names
Nominal Domain Theory

Observations:

1. $\langle a \rangle \bigcup S = \bigcup \{ \langle a \rangle | d \in S\}$ does not hold for all directed (and finitely supported) sets.

2. It does hold for all uniform-directed sets (directed set where all elements are supported by a common finite set).

3. A nominal partial order is finitely-supported-chain complete iff it is uniform-directed complete.

Nominal Scott domain = fs-bounded-complete, $\omega$-algebraic, uniform-directed complete, nominal partial order with bottom.

▶ fs-bounded-complete: finitely supported, bounded subsets have a join
▶ uniform-continuous function: uniform-directed-join preserving function; the finitely supported ones form the nominal Scott domain $D_{uc,fs}$.
Nominal Domain Theory

Observations:

1. \( \langle a \rangle \bigsqcup S = \bigsqcup \{ \langle a \rangle d \mid d \in S \} \) does not hold for all directed (and finitely supported) sets.
Observations:

1. $\langle a \rangle \bigsqcup S = \bigsqcup \{ \langle a \rangle d \mid d \in S \}$ does not hold for all directed (and finitely supported) sets.

2. It does hold for all uniform-directed sets (directed set where all elements are supported by a common finite set)
Nominal Domain Theory

Observations:

1. \( \langle a \rangle \bigsqcup S = \bigsqcup \{ \langle a \rangle d \mid d \in S \} \) does not hold for all directed (and finitely supported) sets.

2. It does hold for all uniform-directed sets (directed set where all elements are supported by a common finite set).

3. A nominal partial order is finitely-supported-chain complete iff it is uniform-directed complete.
Observations:

1. \( \langle a \rangle \bigcup S = \bigcup \{ \langle a \rangle d \mid d \in S \} \) does not hold for all directed (and finitely supported) sets.

2. It does hold for all uniform-directed sets (directed set where all elements are supported by a common finite set).

3. A nominal partial order is finitely-supported-chain complete iff it is uniform-directed complete.

Nominal Scott domain = fs-bounded-complete, \( \omega \)-algebraic, uniform-directed complete, nominal partial order with bottom.
Nominal Domain Theory

Observations:

1. \( \langle a \rangle \bigsqcup S = \bigsqcup \{ \langle a \rangle d \mid d \in S \} \) does not hold for all directed (and finitely supported) sets.

2. It does hold for all uniform-directed sets (directed set where all elements are supported by a common finite set).

3. A nominal partial order is finitely-supported-chain complete iff it is uniform-directed complete.

Nominal Scott domain = fs-bounded-complete, \( \omega \)-algebraic, uniform-directed complete, nominal partial order with bottom.

- **fs-bounded-complete**: finitely supported, bounded subsets have a join
Nominal Domain Theory

Observations:

1. \( \langle a \rangle \bigsqcup S = \bigsqcup \{ \langle a \rangle d \mid d \in S \} \) does not hold for all directed (and finitely supported) sets.

2. It does hold for all uniform-directed sets (directed set where all elements are supported by a common finite set).

3. A nominal partial order is finitely-supported-chain complete iff it is uniform-directed complete.

Nominal Scott domain = fs-bounded-complete, \( \omega \)-algebraic, uniform-directed complete, nominal partial order with bottom.

- **fs-bounded-complete**: finitely supported, bounded subsets have a join
- **uniform-continuous function**: uniform-directed-join preserving function; the finitely supported ones form the nominal Scott domain \( D \rightarrow_{uc,fs} E \)
PNA – Denotational Semantics

Types denote nominal Scott domains:

\( J \\text{bool} \), \( J \\text{nat} \), \( J \tau_1 \times \tau_2 \)

as for PCF

\( J \tau_1 \times \tau_2 = J \tau_1 \text{uc}, \text{fs} J \tau_2 \)

\( J \text{name} = \{a, b, c, \ldots\} \)

\( J \text{term} = (\{t ::= a | t t | \lambda a. t\} / = \alpha) \perp \)

Uniform-continuous name restriction operation:

\( J \text{bool}, J \text{nat}, J \text{name}, J \text{term}: a \\not\in d = \{d \text{if } a \text{ fresh for } d \perp \text{otherwise} \}

\( J \tau_1 \times \tau_2: (a \not\in (d_1, d_2)) = (a \not\in d_1, a \not\in d_2) \)

\( J \tau_1 \tau_2: (a \not\in f d) = a \not\in (f d) \text{ if } a \text{ fresh for } d \)
Types denote nominal Scott domains:
Types denote nominal Scott domains:

- $\llbracket \text{bool} \rrbracket$, $\llbracket \text{nat} \rrbracket$, $\llbracket \tau_1 \times \tau_2 \rrbracket$ as for PCF
Types denote nominal Scott domains:

- $[\text{bool}], [\text{nat}], [\tau_1 \times \tau_2]$ as for PCF
- $[\tau_1 \rightarrow \tau_2] = [\tau_1] \rightarrow_{uc,fs} [\tau_2]$
Types denote nominal Scott domains:

- $[[\text{bool}]]$, $[[\text{nat}]]$, $[[\tau_1 \times \tau_2]]$ as for PCF
- $[[\tau_1 \rightarrow \tau_2]] = [[\tau_1]] \rightarrow_{uc,fs} [[\tau_2]]$
- $[[\text{name}]] = \{a, b, c, \ldots\} \perp$
Types denote nominal Scott domains:

- $[[\text{bool}], [[\text{nat}], [[\tau_1 \times \tau_2]]]$ as for PCF
- $[[\tau_1 \rightarrow \tau_2]] = [[\tau_1]] \rightarrow_{uc,fs} [[\tau_2]]$
- $[[\text{name}]] = \{a, b, c, \ldots\}_\bot$
- $[[\text{term}]] = (\{t ::= a \mid t \mid \lambda a.t\}/=_{\alpha})_\bot$
Types denote nominal Scott domains:

- $[\text{bool}]$, $[\text{nat}]$, $[\tau_1 \times \tau_2]$ as for PCF
- $[\tau_1 \to \tau_2] = [\tau_1] \to_{uc,fs} [\tau_2]$
- $[\text{name}] = \{a, b, c, \ldots\} \bot$
- $[\text{term}] = (\{t ::= a \mid t t \mid \lambda a. t\}/=_{\alpha}) \bot$
- $[\delta \tau] = [\mathcal{A}][\tau]$
Types denote nominal Scott domains:

- $[[\text{bool}]]$, $[[\text{nat}]]$, $[[\tau_1 \times \tau_2]]$ as for PCF
- $[[\tau_1 \rightarrow \tau_2]] = [[\tau_1]] \rightarrow_{uc,fs} [[\tau_2]]$
- $[[\text{name}]] = \{a, b, c, \ldots\}_\perp$
- $[[\text{term}]] = ([t ::= a | t t | \lambda a.t]/=_{\alpha})_\perp$
- $[[\delta \tau]] = [A][\tau]$

Uniform-continuous name restriction operation:
Types denote nominal Scott domains:

- $[[\text{bool}]], [[\text{nat}]], [[\tau_1 \times \tau_2]]$ as for PCF
- $[[\tau_1 \to \tau_2]] = [[\tau_1]] \to_{uc,fs} [[\tau_2]]$
- $[[\text{name}]] = \{a, b, c, \ldots\}_\bot$
- $[[\text{term}]] = (\{t ::= a \mid t t \mid \lambda a.t\}/=\alpha)_\bot$
- $[[\delta \tau]] = [A][\tau]$

Uniform-continuous name restriction operation:

- $[[\text{bool}]], [[\text{nat}]], [[\text{name}]], [[\text{term}]]: a \backslash d = \begin{cases} d & \text{if } a \text{ fresh for } d \\ \bot & \text{otherwise} \end{cases}$
Types denote nominal Scott domains:

- $\text{[bool]}$, $\text{[nat]}$, $\text{[\tau_1 \times \tau_2]}$ as for PCF
- $\text{[\tau_1 \rightarrow \tau_2]} = \text{[\tau_1]} \rightarrow_{uc,fs} \text{[\tau_2]}$
- $\text{[name]} = \{a, b, c, \ldots\}$
- $\text{[term]} = \{t ::= a \mid t \mid \lambda a.t\}/\alpha$
- $\text{[\delta \tau]} = \lceil A \rceil \lceil \tau \rceil$

Uniform-continuous name restriction operation:

- $\text{[bool]}, [\text{nat}], [\text{name}], [\text{term}]: a \setminus d = \begin{cases} \downarrow & \text{if } a \text{ fresh for } d \\ \downarrow & \text{otherwise} \end{cases}$
- $\text{[\tau_1 \times \tau_2]}: a \setminus (d_1, d_2) = (a \setminus d_1, a \setminus d_2)$
Types denote nominal Scott domains:
- $\llbracket \text{bool} \rrbracket$, $\llbracket \text{nat} \rrbracket$, $\llbracket \tau_1 \times \tau_2 \rrbracket$ as for PCF
- $\llbracket \tau_1 \to \tau_2 \rrbracket = [\tau_1] \rightarrow_{uc,fs} [\tau_2]$
- $\llbracket \text{name} \rrbracket = \{a, b, c, \ldots\} \perp$
- $\llbracket \text{term} \rrbracket = (\{t ::= a \mid t t \mid \lambda a.t\}/=_{\alpha}) \perp$
- $\llbracket \delta \tau \rrbracket = [\Lambda][\tau]$

Uniform-continuous name restriction operation:
- $\llbracket \text{bool}, \text{nat}, \text{name}, \text{term} \rrbracket: a \setminus d = \begin{cases} d & \text{if a fresh for } d \\ \perp & \text{otherwise} \end{cases}$
- $\llbracket \tau_1 \times \tau_2 \rrbracket: a \setminus (d_1, d_2) = (a \setminus d_1, a \setminus d_2)$
- $\llbracket \tau_1 \to \tau_2 \rrbracket: (a \setminus f) d = a \setminus (f d) \quad \text{if a fresh for } d$
Types denote nominal Scott domains:

- $\mathbb{[bool]}$, $\mathbb{[nat]}$, $\mathbb{[\tau_1 \times \tau_2]}$ as for PCF
- $\mathbb{[\tau_1 \rightarrow \tau_2]} = \mathbb{[\tau_1]} \rightarrow_{uc,fs} \mathbb{[\tau_2]}$
- $\mathbb{[name]} = \{a, b, c, \ldots\}_\bot$
- $\mathbb{[\text{term}]} = (\{t ::= a \mid t \; t \mid \lambda a. t\}/=\alpha)_\bot$
- $\mathbb{[\delta \tau]} = [A][\tau]$

Uniform-continuous name restriction operation:

- $\mathbb{[bool]}, [\mathbb{[nat]}], [\mathbb{[name]}], [\mathbb{[term]}]: a \\setminus d = \begin{cases} d & \text{if } a \text{ fresh for } d \\ \bot & \text{otherwise} \end{cases}$
- $\mathbb{[\tau_1 \times \tau_2]}: a \setminus (d_1, d_2) = (a \setminus d_1, a \setminus d_2)$
- $\mathbb{[\tau_1 \rightarrow \tau_2]}: (a \setminus f) d = a \setminus (f \; d) \quad \text{if } a \text{ fresh for } d$
- $\mathbb{[\delta \tau]}: a \setminus (\langle a' \rangle d) = \langle a' \rangle (a \setminus d) \quad \text{if } a \neq a'$
Expressions denote uniform-continuous, finitely supp. functions:
Expressions denote uniform-continuous, finitely supp. functions:

- $\llbracket a \rrbracket \rho = a$
Expressions denote uniform-continuous, finitely supp. functions:

- $[a] \rho = a$
- $[\nu a. e] \rho = a \setminus ([e] \rho)$ if $a \not\# \rho$
Expressions denote uniform-continuous, finitely supp. functions:

- $\llbracket a \rrbracket \rho = a$
- $\llbracket \nu a. e \rrbracket \rho = a \setminus (\llbracket e \rrbracket \rho)$ if $a \not\equiv \rho$
- $\llbracket \alpha a. e \rrbracket \rho = \langle a \rangle (\llbracket e \rrbracket \rho)$ if $a \not\equiv \rho$
Expressions denote uniform-continuous, finitely supp. functions:

- $\llbracket a \rrbracket \rho = a$
- $\llbracket \nu a. e \rrbracket \rho = a \setminus (\llbracket e \rrbracket \rho)$ if $a \neq \rho$
- $\llbracket \alpha a. e \rrbracket \rho = \langle a \rangle (\llbracket e \rrbracket \rho)$ if $a \neq \rho$
- $\llbracket e_1 \& e_2 \rrbracket \rho = \begin{cases} d & \text{if } \llbracket e_1 \rrbracket \rho = \langle a \rangle d \land \llbracket e_2 \rrbracket \rho = a \\ a \setminus (a a') \cdot d & \text{if } \llbracket e_1 \rrbracket \rho = \langle a \rangle d \land \llbracket e_2 \rrbracket \rho = a' \neq a \\ \bot & \text{otherwise} \end{cases}$
Expressions denote uniform-continuous, finitely supp. functions:

- $[[a]]\rho = a$
- $[[\nu a. e]]\rho = a \setminus (\llbracket e \rrbracket \rho)$ if $a \neq \rho$
- $[[\alpha a. e]]\rho = \langle a \rangle (\llbracket e \rrbracket \rho)$ if $a \neq \rho$
- $[[e_1 @ e_2]]\rho = \begin{cases} 
  d & \text{if } \llbracket e_1 \rrbracket \rho = \langle a \rangle d \land \llbracket e_2 \rrbracket \rho = a \\
  a \setminus (a a') \cdot d & \text{if } \llbracket e_1 \rrbracket \rho = \langle a \rangle d \land \llbracket e_2 \rrbracket \rho = a' \neq a \\
  \perp & \text{otherwise}
\end{cases}$
- $[[\langle e_1 \Rightarrow e_2 \rangle e_3]]\rho = \begin{cases} 
  (a_1 a_2) \cdot (\llbracket e_3 \rrbracket \rho) & \text{if } \llbracket e_i \rrbracket \rho = a_i \ (i = 1, 2) \\
  \perp & \text{otherwise}
\end{cases}$
Expressions denote uniform-continuous, finitely supp. functions:

- $\llbracket a \rrbracket \rho = a$
- $\llbracket \nu a. e \rrbracket \rho = a \setminus (\llbracket e \rrbracket \rho)$ if $a \neq \rho$
- $\llbracket \alpha a. e \rrbracket \rho = \langle a \rangle (\llbracket e \rrbracket \rho)$ if $a \neq \rho$
- $\llbracket e_1 \odot e_2 \rrbracket \rho = \begin{cases} d & \text{if } \llbracket e_1 \rrbracket \rho = \langle a \rangle d \land \llbracket e_2 \rrbracket \rho = a \\ a \setminus (a a') \cdot d & \text{if } \llbracket e_1 \rrbracket \rho = \langle a \rangle d \land \llbracket e_2 \rrbracket \rho = a' \neq a \\ \bot & \text{otherwise} \end{cases}$
- $\llbracket (e_1 \Rightarrow e_2) e_3 \rrbracket \rho = \begin{cases} (a_1 a_2) \cdot (\llbracket e_3 \rrbracket \rho) & \text{if } \llbracket e_i \rrbracket \rho = a_i \ (i = 1, 2) \\ \bot & \text{otherwise} \end{cases}$
- $\llbracket e_1 = e_2 \rrbracket \rho = \begin{cases} \text{true} & \text{if } \llbracket e_i \rrbracket \rho = a \ (i = 1, 2) \\ \text{false} & \text{if } \bot \neq \llbracket e_1 \rrbracket \rho \neq \llbracket e_2 \rrbracket \neq \bot \\ \bot & \text{otherwise} \end{cases}$
This denotational semantics is adequate: $Je_1K\rho = Je_2K\rho \Rightarrow e_1 \sim e_2$
PNA – Denotational Semantics (3)

\[
[V \ e] \rho = \begin{cases} 
[a]_\alpha & \text{if } [e] \rho = a \\
\bot & \text{otherwise}
\end{cases}
\]

\[
[A \ e_1 \ e_2] \rho = \begin{cases} 
[t_1 \ t_2]_\alpha & \text{if } [e_i] \rho = [t_i]_\alpha \ (i = 1, 2) \\
\bot & \text{otherwise}
\end{cases}
\]

\[
[L \ e] \rho = \begin{cases} 
[\lambda a. t]_\alpha & \text{if } [e] \rho = \langle a \rangle [t]_\alpha \\
\bot & \text{otherwise}
\end{cases}
\]
PNA – Denotational Semantics (3)

▷ \[[V \, e] \rho = \begin{cases} 
[a]_\alpha & \text{if } [e] \rho = a \\
\bot & \text{otherwise}
\end{cases} \]

▷ \[[A \, e_1 \, e_2] \rho = \begin{cases} 
[t_1 \, t_2]_\alpha & \text{if } [e_i] \rho = [t_i]_\alpha \ (i = 1, 2) \\
\bot & \text{otherwise}
\end{cases} \]

▷ \[[L \, e] \rho = \begin{cases} 
[\lambda a. \, t]_\alpha & \text{if } [e] \rho = \langle a \rangle [t]_\alpha \\
\bot & \text{otherwise}
\end{cases} \]

▷ \[[\text{case } e \, \text{of } (V \, x_1 \rightarrow e_1 \mid A \, x_2 \, x_2' \rightarrow e_2 \mid L \, x_3 \rightarrow e_3)] \rho = \begin{cases} 
[e_1] \rho[x_1 \mapsto a] & \text{if } [e] \rho = [a]_\alpha \\
[e_2] \rho[x_2 \mapsto [t]_\alpha, x_2' \mapsto [t']_\alpha] & \text{if } [e] \rho = [t \, t']_\alpha \\
[e_3] \rho[x_3 \mapsto \langle a \rangle [t]_\alpha] & \text{if } [e] \rho = [\lambda a. \, t]_\alpha \\
\bot & \text{otherwise}
\end{cases} \]
This denotational semantics is **adequate**: 

\[
\llbracket e_1 \rrbracket = \llbracket e_2 \rrbracket \Rightarrow e_1 \cong_{\text{PNA}} e_2
\]
PNA+ = PNA + parallel-or + name-exists + definite-description

Syntax:
\[ e ::= \ldots | e \lor e | ex \cdot e | \text{the} \cdot e \]

Typing:
\[ \Gamma, x : \text{name} \vdash e : \text{bool} \]
\[ \Gamma \vdash ex \cdot e : \text{bool} \]
\[ \Gamma, x : \text{name} \vdash e : \text{bool} \]
\[ \Gamma \vdash the \cdot e : \text{name} \]

Operational semantics:
\[ e[a/x] \Downarrow T \]
\[ ex \cdot e \Downarrow T \]
\[ e[a/x] \Downarrow F \]
\[ \exists b \in \text{fn}(e) \cup \{a\}' \]
\[ e[b/x] \Downarrow F \]
\[ \exists b \in \text{fn}(e) \setminus \{a\} \cup \{a\}' \]

Steffen Lösch and Andrew M. Pitts
PNA+ = PNA + parallel-or + name-exists + definite-description
PNA+

PNA+ = PNA + parallel-or + name-exists + definite-description

- **Syntax**: \( e ::= \ldots | e \lor e | \text{ex} \, x \, . \, e | \text{the} \, x \, . \, e \)
PNA+ = PNA + parallel-or + name-exists + definite-description

- Syntax: \[ e ::= \ldots | e \text{ por } e | \text{ex } x. e | \text{the } x. e \]

- Typing:
  \[
  \frac{\Gamma, x : \text{name} \vdash e : \text{bool}}{\Gamma \vdash \text{ex } x. e : \text{bool}} \quad \frac{\Gamma, x : \text{name} \vdash e : \text{bool}}{\Gamma \vdash \text{the } x. e : \text{name}}
  \]
PNA+ = PNA + parallel-or + name-exists + definite-description

- **Syntax:** $e ::= \ldots | e \text{ por } e | \text{ex } x. e | \text{the } x. e$

- **Typing:**
  
  \[
  \frac{}{\Gamma, x : \text{name} \vdash e : \text{bool}} \quad \frac{}{\Gamma, x : \text{name} \vdash e : \text{bool}}
  \]

  \[
  \frac{}{\Gamma \vdash \text{ex } x. e : \text{bool}} \quad \frac{}{\Gamma \vdash \text{the } x. e : \text{name}}
  \]

- **Operational semantics:**

  \[
  \frac{e[a/x] \Downarrow T}{\text{ex } x. e \Downarrow T}
  \quad \frac{a' \not\# e}{(\forall b \in \text{fn}(e) \cup \{a'\}) \ e[b/x] \Downarrow \text{F}}
  \]

  \[
  \frac{a' \not\# (e, a)}{(\forall b \in (\text{fn}(e) - \{a\}) \cup \{a'\}) \ e[b/x] \Downarrow \text{F}}
  \]

  \[
  \frac{e[a/x] \Downarrow T \quad (\forall b \in (\text{fn}(e) - \{a\}) \cup \{a'\}) \ e[b/x] \Downarrow \text{F}}{\text{the } x. e \Downarrow a}
  \]
Full Abstraction for PNA+

- Denotational semantics:
Denotational semantics:

\[
\left[ \text{ex } x. \ e \right] \rho = \begin{cases} 
    \text{true} & \text{if } (\exists a \in A) \left[ e \right](\rho[x \mapsto a]) = \text{true} \\
    \text{false} & \text{if } (\forall a \in A) \left[ e \right](\rho[x \mapsto a]) = \text{false} \\
    \bot & \text{otherwise}
\end{cases}
\]
Full Abstraction for PNA+

- **Denotational semantics:**

\[
\begin{align*}
\llbracket \exists x. e \rrbracket \rho &= \begin{cases} 
  \text{true} & \text{if } (\exists a \in A) \llbracket e \rrbracket (\rho[x \mapsto a]) = \text{true} \\
  \text{false} & \text{if } (\forall a \in A) \llbracket e \rrbracket (\rho[x \mapsto a]) = \text{false} \\
  \bot & \text{otherwise}
\end{cases} \\
\llbracket \text{the } x. e \rrbracket \rho &= \begin{cases} 
  a & \text{if } \llbracket \lambda x : \text{name} \to e \rrbracket \rho = \text{eq}_a \\
  \bot & \text{otherwise}
\end{cases}
\end{align*}
\]
Full Abstraction for PNA+

- *Denotational semantics:*

\[
\begin{aligned}
\lbrack \text{ex } x. \ e \rbrack \rho &= \begin{cases} 
  \text{true} & \text{if } (\exists a \in A) \ [e](\rho[x \mapsto a]) = \text{true} \\
  \text{false} & \text{if } (\forall a \in A) \ [e](\rho[x \mapsto a]) = \text{false} \\
  \bot & \text{otherwise}
\end{cases} \\
\lbrack \text{the } x. \ e \rbrack \rho &= \begin{cases} 
  a & \text{if } [\lambda x: \text{name} \to e] \rho = eq_a \\
  \bot & \text{otherwise}
\end{cases}
\end{aligned}
\]

PNA+ is fully abstract:

\[
\lbrack e_1 \rbrack = \lbrack e_2 \rbrack \iff e_1 \equiv \text{PNA+} e_2.
\]
Full Abstraction for PNA+

**Denotational semantics:**

\[
\begin{align*}
\llbracket \text{ex } x. \ e \rrbracket \rho &= \begin{cases} 
\text{true} & \text{if } (\exists a \in A) \ \llbracket e \rrbracket (\rho[x \mapsto a]) = \text{true} \\
\text{false} & \text{if } (\forall a \in A) \ \llbracket e \rrbracket (\rho[x \mapsto a]) = \text{false} \\
\bot & \text{otherwise}
\end{cases} \\
\llbracket \text{the } x. \ e \rrbracket \rho &= \begin{cases} 
\ a & \text{if } \llbracket \lambda x : \text{name} \to e \rrbracket \rho = eq_a \\
\bot & \text{otherwise}
\end{cases}
\end{align*}
\]

PNA+ is fully abstract:

\[
\llbracket e_1 \rrbracket = \llbracket e_2 \rrbracket \iff e_1 \cong_{\text{PNA+}} e_2.
\]

Proof works again by compact definability, but only at a subset of all types. We use *definable retracts* to finish the proof.
Open Problems

1. Are all compact elements definable in PNA+?
2. Does full abstraction fail for PNA+por, PNA+por+ex or PNA+por+the?
3. Is there a fully abstract game semantics for PNA?
4. Nominal information systems? Nominal powerdomains?
Open Problems

1. Are all compact elements definable in PNA+?
Open Problems

1. Are all compact elements definable in PNA+?

2. Does full abstraction fail for PNA+\(\text{por}\), PNA+\(\text{por+ex}\) or PNA+\(\text{por+the}\)?
Open Problems

1. Are all compact elements definable in PNA+?

2. Does full abstraction fail for PNA+\textit{por}, PNA+\textit{por}+\textit{ex} or PNA+\textit{por}+\textit{the}?

3. Is there a fully abstract game semantics for PNA?
Open Problems

1. Are all compact elements definable in PNA+?

2. Does full abstraction fail for PNA+\text{por}, PNA+\text{por+ex} or PNA+\text{por+the}?

3. Is there a fully abstract game semantics for PNA?

4. Nominal information systems? Nominal powerdomains?
Conclusions

Steffen Lösch and Andrew M. Pitts

Full Abstraction for PCF with Names
Conclusions

- **meta-level**
- **object-level**

---

PNA

- booleans, numbers, $\lambda$-calculus

▶ used nominal sets to enhance a semantic theory
▶ obtained a more expressive language
▶ retrieved classical results

complications from nominal sets were somehow orthogonal
Conclusions

\begin{itemize}
  \item \textit{meta-level}
  \item \textit{object-level}
  \item PNA
    \begin{itemize}
      \item booleans, numbers, $\lambda$-calculus
    \end{itemize}
\end{itemize}

\begin{itemize}
  \item adequacy
  \item operational semantics $\approx$
  \item denotational semantics $=\ $
\end{itemize}

full abstraction

Steffen Lösch and Andrew M. Pitts

Full Abstraction for PCF with Names
Conclusions

- meta-level
- object-level

PNA

booleans, numbers, \( \lambda \)-calculus

adequacy

operational semantics \( \simeq \) \[ \] denotational semantics \( = \)

full abstraction

- used nominal sets to enhance a semantic theory
Conclusions

- **meta-level**
- **object-level**
- **PNA**

| booleans, numbers, $\lambda$-calculus |

- adequacy

- operational semantics $\simeq$
- denotational semantics $=$

- full abstraction

- used nominal sets to enhance a semantic theory
- obtained a more expressive language
Conclusions

- used nominal sets to enhance a semantic theory
- obtained a more expressive language
- retrieved classical results
Conclusions

- Used nominal sets to enhance a semantic theory
- Obtained a more expressive language
- Retrieved classical results
- Complications from nominal sets were somehow orthogonal
Backup Slides
Orbit-Finite Subsets

A finitely supported subset $S \in P_{fs}X$ of a nominal set $X$ is orbit-finite iff it is contained in finitely many orbits of $X$. An orbit $x$ and $x'$ are in the same orbit iff $x' = \pi \cdot x$ for some $\pi$.

Theorem:
For any nominal set $X$, the uniform-compact elements of $P_{fs}X$ is uniform-compact if and only if they are orbit-finite. A subset is orbit-finite iff it is of the form $\text{hull } A F$.

So there is the following analogy:
finite directed sets $\sim$ orbit-finite uniform-directed nominal sets.
A finitely supported subset $S \in \mathcal{P}_{fs}X$ of a nominal set $X$ is **orbit-finite** iff it is contained in finitely many orbits of $X$. 
A finitely supported subset $S \in \mathcal{P}_{fs}X$ of a nominal set $X$ is **orbit-finite** iff it is contained in finitely many orbits of $X$.

- **orbit**: $x$ and $x'$ are in the same orbit iff $x' = \pi \cdot x$ for some $\pi$.
A finitely supported subset $S \in P_{fs}X$ of a nominal set $X$ is **orbit-finite** iff it is contained in finitely many orbits of $X$

- **orbit**: $x$ and $x'$ are in the same orbit iff $x' = \pi \cdot x$ for some $\pi$
- **hull**: $\text{hull}_A F = \{\pi \cdot x \mid \pi \not\# A \land x \in F\}$, where $A \subseteq_f A$ and $F \subseteq_f X$
A finitely supported subset $S \in P_{fs}X$ of a nominal set $X$ is **orbit-finite** iff it is contained in finitely many orbits of $X$

- **orbit**: $x$ and $x'$ are in the same orbit iff $x' = \pi \cdot x$ for some $\pi$
- **hull**: $\text{hull}_A F = \{ \pi \cdot x \mid \pi \not\in A \land x \in F \}$, where $A \subseteq_f A$ and $F \subseteq_f X$

**Theorem:**

For any nominal set $X$, the uniform-compact elements of $P_{fs}X$ is uniform-compact if are exactly the orbit-finite subset of $X$. A subset is orbit-finite iff it is of the form $\text{hull}_A F$. 
Orbit-Finite Subsets

A finitely supported subset $S \in P_{fs}X$ of a nominal set $X$ is **orbit-finite** iff it is contained in finitely many orbits of $X$

- **orbit**: $x$ and $x'$ are in the same orbit iff $x' = \pi \cdot x$ for some $\pi$
- **hull**: $\text{hull}_A F = \{ \pi \cdot x | \pi \# A \land x \in F \}$, where $A \subseteq_f A$ and $F \subseteq_f X$

**Theorem:**

For any nominal set $X$, the uniform-compact elements of $P_{fs}X$ is uniform-compact if are exactly the orbit-finite subset of $X$. A subset is orbit-finite iff it is of the form $\text{hull}_A F$.

So there is the following analogy

finite directed sets $\sim$ orbit-finite uniform-directed nominal sets

Steffen Lösch and Andrew M. Pitts  Full Abstraction for PCF with Names
Typing judgement: $\Gamma \vdash e : \tau$

- $c = T \mid F$
  $\Gamma \vdash c : \text{bool}$

- $\Gamma \vdash e : \text{nat}$
  $\Gamma \vdash \text{zero} \; e : \text{bool}$
  $\Gamma \vdash e : \tau_1 \times \tau_2$
  $\Gamma \vdash \text{snd} \; e : \tau_2$

- $\Gamma \vdash e_1 : \tau_1$
  $\Gamma \vdash (e_1, e_2) : \tau_1 \times \tau_2$
  $\Gamma \vdash \text{fst} \; e : \tau_1$

- $\Gamma \vdash e_2 : \tau$
  $\Gamma, x : \tau \vdash e : \tau'$

- $\Gamma \vdash e_1 : \tau \rightarrow \tau'$
  $\Gamma \vdash e_2 : \tau$
  $\Gamma \vdash e_1 \; e_2 : \tau'$

- $\Gamma \vdash \lambda x : \tau \rightarrow e : \tau ightarrow \tau'$

- $\Gamma \vdash \text{fix} \; e : \tau$
  $\Gamma \vdash 0 : \text{nat}$

- $\Gamma \vdash e : \text{nat}$
  $\Gamma \vdash S \; e : \text{nat}$
  $\Gamma \vdash \text{pred} \; e : \text{nat}$

$$\Gamma = \{x_1 : \tau_1, \ldots, x_n : \tau_n\}$$

Steffen Lösch and Andrew M. Pitts
Full Abstraction for PCF with Names
PCF – Full Operational Semantics

Evaluation judgement: \( e \Downarrow c \)

\[ c \in \text{Can} ::= T \mid F \mid O \mid S \mid (e, e) \mid \lambda x : \tau \to e \]

closed expression

\[
\begin{align*}
    c &= T \mid F \mid 0 \mid (e_1, e_2) \mid \lambda x : \tau \to e \\
    \frac{c \Downarrow c}{e \Downarrow c} &\quad \frac{S e \Downarrow S c}{e \Downarrow S c} &\quad \frac{\text{pred } e \Downarrow c}{e \Downarrow 0} \\
    \frac{e_1 \Downarrow T \quad e_2 \Downarrow c}{\text{if } e_1 \text{ then } e_2 \text{ else } e_3 \Downarrow c} &\quad \frac{e_1 \Downarrow F \quad e_3 \Downarrow c}{\text{if } e_1 \text{ then } e_2 \text{ else } e_3 \Downarrow c} &\quad \frac{\text{zero } e \Downarrow T}{e \Downarrow 0} \\
    \frac{e \Downarrow S c}{\text{zero } e \Downarrow F} &\quad \frac{e \Downarrow (e_1, e_2) \quad e_1 \Downarrow c}{\text{fst } e \Downarrow c} &\quad \frac{e \Downarrow (e_1, e_2) \quad e_2 \Downarrow c}{\text{snd } e \Downarrow c} \\
    \frac{e_1 \Downarrow \lambda x : \tau \to e \quad e[e_2/x] \Downarrow c}{e_1 \ e_2 \Downarrow c} &\quad \frac{e(\text{fix } e) \Downarrow c}{\text{fix } e \Downarrow c}
\end{align*}
\]
PCF – Full Denotational Semantics

\[ \Gamma \vdash e : \tau \Rightarrow [e] \in [\Gamma] \rightarrow_c [\tau] \]

Types denote domains:
- \([\text{bool}] = \{\text{true, false}\}_\perp\)
- \([\text{nat}] = \{0, 1, 2, \ldots\}_\perp\)
- \([\tau_1 \times \tau_2] = [\tau_1] \times [\tau_2]\)
- \([\tau_1 \rightarrow \tau_2] = [\tau_1] \rightarrow_c [\tau_2]\)
- \([\Gamma] = [\tau_1] \times \ldots \times [\tau_n]\) if \(\Gamma = \{x_1 : \tau_1, \ldots, x_n : \tau_n\}\)

Expressions denote continuous functions:
- \([x]_\rho = \rho \cdot x\)
- \([\text{T}]_\rho = \text{true}\)
- \([\text{F}]_\rho = \text{false}\)
- \([\text{if } e_1 \text{ then } e_2 \text{ else } e_3]_\rho = \begin{cases} [e_2]_\rho & \text{if } [e_1]_\rho = \text{true} \\ [e_3]_\rho & \text{if } [e_1]_\rho = \text{false} \\ \bot & \text{otherwise} \end{cases}\)
PCF – Full Denotational Semantics (2)

- \([0]_\rho = 0\)
- \([S \ e]_\rho = \begin{cases} n + 1 & \text{if } [e]_\rho = n \in \mathbb{N} \\ \bot & \text{otherwise} \end{cases}\)
- \([\text{pred} \ e]_\rho = \begin{cases} n & \text{if } [e]_\rho = n + 1 \in \mathbb{N} \\ \bot & \text{otherwise} \end{cases}\)
- \([\text{zero} \ e]_\rho = \begin{cases} \text{true} & \text{if } [e]_\rho = 0 \in \mathbb{N} \\ \text{false} & \text{if } [e]_\rho = n + 1 \in \mathbb{N} \\ \bot & \text{otherwise} \end{cases}\)
- \([(e_1 , e_2)]_\rho = ([e_1]_\rho , [e_2]_\rho)\)
- \([\text{fst} \ e]_\rho = \pi_1([e]_\rho)\\ [\text{snd} \ e]_\rho = \pi_2([e]_\rho)\)
- \([\lambda x : \tau \rightarrow e]_\rho = \lambda \ d \in [\tau] . [e]_\rho([x \mapsto d])\)
- \([e_1 \ e_2]_\rho = [e_1]_\rho ([e_2]_\rho)\)
- \([\text{fix} \ e]_\rho = \text{fix}([e]_\rho)\)