The Safe $\lambda$-Calculus

William Blum

Joint work with C.-H. Luke Ong

Oxford University Computing Laboratory

Lunch-time meeting, 14 May 2007
Overview

- **Safety** is originally a syntactic restriction for higher-order grammars with nice automata-theoretic characterization.
- In the context of the $\lambda$-calculus it gives rise to the **Safe $\lambda$-calculus**.
- The loss of expressivity can be characterized in terms of representable numeric functions.
- The calculus has a “succinct” game-semantic model.
Outline for this talk

Part I The safety restriction
  1. Safety for higher-order grammars
  2. The safe $\lambda$-calculus
  3. Expressivity

Part II Game-semantic
  1. The Correspondence Theorem
  2. Game-semantic characterisation
  3. Compositionality
Part I : The Safety Restriction
Higher-order grammars

Notation for types: $A_1 \rightarrow (A_2 \rightarrow (\ldots (A_n \rightarrow o))\ldots)$ is written $(A_1, A_2, \ldots, A_n, o)$.

▶ Higher-order grammars (Maslov, 1974) are used as generators of word languages, trees or graphs.
▶ A higher-grammar is formally given by a tuple $\langle \Sigma, \mathcal{N}, \mathcal{R}, S \rangle$ (terminals, non-terminals, rewriting rules, starting symbol).
▶ Example of a tree-generating order-2 grammar:

\[
\begin{align*}
S & \rightarrow Ha \\
Hz^o & \rightarrow F(gz) \\
F \phi^{(o,o)} & \rightarrow \phi(\phi(Fh))
\end{align*}
\]

Non-terminals: $S : o$, $H : (o, o)$ and $F : ((o, o), o)$. Terminals: $a : o$ and $g, h : (o, o)$. 
The Safety Restriction

- First appeared under the name “restriction of derived types” in “IO and OI Hierarchies” by W. Damm, TCS 1982
- It is a syntactic restriction for higher-order grammars that constrains the occurrences of the variables in the grammar equations according to their orders.

$(A_1, \ldots, A_n, o)$ is homogeneous if $A_1$, $\ldots$, $A_n$ are and $\text{ord } A_1 \geq \text{ord } A_2 \geq \cdots \geq \text{ord } A_n$.

Definition (Knapik, Niwiński and Urzyczyn (2001-2002))

All types are assumed to be homogeneous.
An order $k > 0$ term is unsafe if it contains an occurrence of a parameter of order strictly less than $k$. An unsafe subterm $t$ of $t'$ occurs in safe position if it is in operator position ($t' = \cdots (ts) \cdots$). A grammar is safe if at the right-hand side of any production all unsafe subterms occur in safe positions.
The Safety Restriction

- First appeared under the name “restriction of derived types” in “IO and OI Hierarchies” by W. Damm, TCS 1982
- It is a **syntactic restriction** for higher-order grammars that constrains the occurrences of the variables in the grammar equations according to their orders.
- \((A_1, \cdots, A_n, o)\) is **homogeneous** if \(A_1, \ldots, A_n\) are and \(\text{ord } A_1 \geq \text{ord } A_2 \geq \cdots \geq \text{ord } A_n\).

**Definition (Knapik, Niwiński and Urzyczyn (2001-2002))**

All types are assumed to be **homogeneous**.
An order \(k > 0\) term is **unsafe** if it contains an occurrence of a parameter of order strictly less than \(k\). An unsafe subterm \(t\) of \(t'\) occurs in **safe position** if it is in operator position \((t' = \cdots (ts) \cdots)\). A grammar is **safe** if at the right-hand side of any production all unsafe subterms occur in safe positions.
Some Results On Safety

Damm82 For generating word languages, order-$n$ safe grammars are equivalent to order-$n$ pushdown automata.

KNU02 Generalization of Damm’s result to tree generating safe grammars/PDAs.

KNU02 The Monadic Second Order (MSO) model checking problem for trees generated by safe higher-order grammars of any order is decidable.

Ong06 But anyway, KNU02 result’s is also true for unsafe grammars...

Caucal02 Graphs generated by safe grammars have a decidable MSO theory.

HMOS06 Caucl’s result does not extend to unsafe grammars. However deciding $\mu$-calculus theories is $n$-EXPTIME complete.

AdMO04 Proposed a notion of safety for the $\lambda$-calculus (unpublished).
Simply Typed $\lambda$-Calculus

- Simple types $A := o \mid A \to A$.

- The order of a type is given by $\text{order}(o) = 0$, $\text{order}(A \to B) = \max(\text{order}(A) + 1, \text{order}(B))$.

- Judgements of the form $\Gamma \vdash M : T$ where $\Gamma$ is the context, $M$ is the term and $T$ is the type:

  \[
  \frac{\text{(var)}}{x : A \vdash x : A}
  \quad \frac{\text{(wk)}}{\Delta \vdash M : A \quad \Gamma \subset \Delta}
  \]

  \[
  \frac{\text{(app)}}{\Gamma \vdash M : A \to B \quad \Gamma \vdash N : A}
  \quad \frac{\text{(abs)}}{\Gamma \vdash \lambda x^A.M : A \to B}
  \]

- Example: $f : o \to o \to o, x : o \vdash (\lambda \varphi^o \to o . x^o . \varphi . x)(f \ x)$

- A single rule: $\beta$-reduction. e.g. $(\lambda x.M)N \to_\beta M[N/x]$
The Safe $\lambda$-Calculus

The formation rules

$(\text{var})$ \[ x : A \vdash_s x : A \]

$(\text{wk})$ \[ \frac{\Gamma \vdash_s M : A}{\Delta \vdash_s M : A} \quad \Gamma \subset \Delta \]

$(\text{app})$ \[ \frac{\Gamma \vdash M : (A_1, \ldots, A_i, B) \quad \Gamma \vdash_s N_1 : A_1 \quad \ldots \quad \Gamma \vdash_s N_i : A_i}{\Gamma \vdash_s MN_1 \ldots N_i : B} \]

with the side-condition $\forall y \in \Gamma : \text{ord } y \geq \text{ord } B$

$(\text{abs})$ \[ \frac{\Gamma, x_1 : A_1 \ldots x_n : A_n \vdash_s M : B}{\Gamma \vdash_s \lambda x_1 : A_1 \ldots x_n : A_n. M : A_1 \to \ldots \to A_n \to B} \]

with the side-condition $\forall y \in \Gamma : \text{ord } y \geq \text{ord } A_1 \to \ldots \to A_n \to B$

Lemma

If $\Gamma \vdash_s M : A$ then every free variable in $M$ has order at least $\text{ord } A$.
Variable Capture

The usual “problem” in \(\lambda\)-calculus: avoid variable capture when performing substitution: \((\lambda x. (\lambda y. x)) y \rightarrow_{\beta} (\lambda y. x)[y/x] \neq \lambda y. y\)

1. Standard solution: Barendregt’s convention. Variables are renamed so that free variables and bound variables have different names. Eg. \((\lambda x. (\lambda y. x)) y\) becomes \((\lambda x. (\lambda z. x)) y\) which reduces to \((\lambda z. x)[y/x] = \lambda z. y\)
   
   Drawback: requires to have access to an unbounded supply of names to perform a given sequence of \(\beta\)-reductions.

2. Another solution: use the \(\lambda\)-calculus à la de Brujin where variable binding is specified by an index instead of a name. Variable renaming then becomes unnecessary.
   
   Drawback: the conversion to nameless de Brujin \(\lambda\)-terms requires an unbounded supply of indices.

Property

In the Safe \(\lambda\)-calculus there is no need to rename variables when performing substitution.
Variable Capture

The usual "problem" in $\lambda$-calculus: avoid variable capture when performing substitution: $(\lambda x.(\lambda y.x))y \rightarrow_\beta (\lambda y.x)[y/x] \neq \lambda y.y$

1. **Standard solution**: Barendregt’s convention. Variables are renamed so that free variables and bound variables have different names. Eg. $(\lambda x.(\lambda y.x))y$ becomes $(\lambda x.(\lambda z.x))y$ which reduces to $(\lambda z.x)[y/x] = \lambda z.y$

   **Drawback**: requires to have access to an unbounded supply of names to perform a given sequence of $\beta$-reductions.

2. **Another solution**: use the $\lambda$-calculus à la de Bruijn where variable binding is specified by an index instead of a name. Variable renaming then becomes unnecessary.

   **Drawback**: the conversion to nameless de Bruijn $\lambda$-terms requires an unbounded supply of indices.

Property

In the Safe $\lambda$-calculus there is no need to rename variables when performing substitution.
Variable Capture

The usual “problem” in λ-calculus: avoid variable capture when performing substitution: \((\lambda x. (\lambda y. x)) y \rightarrow_\beta (\lambda y. x)[y/x] \neq \lambda y. y\)

1. **Standard solution**: Barendregt’s convention. Variables are renamed so that free variables and bound variables have different names. Eg. \((\lambda x. (\lambda y. x)) y\) becomes \((\lambda x. (\lambda z. x)) y\) which reduces to \((\lambda z. x)[y/x] = \lambda z. y\)

   **Drawback**: requires to have access to an unbounded supply of names to perform a given sequence of \(\beta\)-reductions.

2. **Another solution**: use the λ-calculus à la de Bruijn where variable binding is specified by an index instead of a name. Variable renaming then becomes unnecessary.

   **Drawback**: the conversion to nameless de Bruijn λ-terms requires an unbounded supply of indices.

**Property**

In the Safe λ-calculus there is no need to rename variables when performing substitution.
Variable Capture

The usual “problem” in λ-calculus: avoid variable capture when performing substitution: \((\lambda x.(\lambda y.x))y \rightarrow_\beta (\lambda y.x)[y/x] \neq \lambda y.y\)

1. Standard solution: Barendregt’s convention. Variables are renamed so that free variables and bound variables have different names. Eg. \((\lambda x.(\lambda y.x))y\) becomes \((\lambda x.(\lambda z.x))y\) which reduces to \((\lambda z.x)[y/x] = \lambda z.y\)

**Drawback:** requires to have access to an unbounded supply of names to perform a given sequence of \(\beta\)-reductions.

2. Another solution: use the λ-calculus à la de Bruijn where variable binding is specified by an index instead of a name. Variable renaming then becomes unnecessary.

**Drawback:** the conversion to nameless de Bruijn λ-terms requires an unbounded supply of indices.

**Property**

In the Safe λ-calculus there is no need to rename variables when performing substitution.
Variable Capture

The usual “problem” in \( \lambda \)-calculus: avoid variable capture when performing substitution: \( (\lambda x.(\lambda y.x))y \rightarrow_\beta (\lambda y.x)[y/x] \neq \lambda y.y \)

1. **Standard solution**: Barendregt’s convention. Variables are renamed so that free variables and bound variables have different names. Eg. \( (\lambda x.(\lambda y.x))y \) becomes \( (\lambda x.(\lambda z.x))y \) which reduces to \( (\lambda z.x)[y/x] = \lambda z.y \)

   **Drawback**: requires to have access to an unbounded supply of names to perform a given sequence of \( \beta \)-reductions.

2. **Another solution**: use the \( \lambda \)-calculus à la de Brujin where variable binding is specified by an index instead of a name. Variable renaming then becomes unnecessary.

   **Drawback**: the conversion to nameless de Brujin \( \lambda \)-terms requires an unbounded supply of indices.

Property

In the Safe \( \lambda \)-calculus there is no need to rename variables when performing substitution.
Variable Capture

The usual “problem” in \( \lambda \)-calculus: avoid variable capture when performing substitution: 
\[
(\lambda x.(\lambda y.x))y \rightarrow_\beta (\lambda y.x)[y/x] \neq \lambda y.y
\]

1. **Standard solution**: Barendregt’s convention. Variables are renamed so that free variables and bound variables have different names. Eg. 
\[
(\lambda x.(\lambda y.x))y \text{ becomes } (\lambda x.(\lambda z.x))y
\]
which reduces to 
\[
(\lambda z.x)[y/x] = \lambda z.y
\]
**Drawback**: requires to have access to an unbounded supply of names to perform a given sequence of \( \beta \)-reductions.

2. **Another solution**: use the \( \lambda \)-calculus à la de Brujin where variable binding is specified by an index instead of a name. Variable renaming then becomes unnecessary.
**Drawback**: the conversion to nameless de Brujin \( \lambda \)-terms requires an unbounded supply of indices.

**Property**

In the Safe \( \lambda \)-calculus there is no need to rename variables when performing substitution.
Examples

1. Contracting the $\beta$-redex in the following term

   \[
   f : o \rightarrow o \rightarrow o, \ x : o \vdash (\lambda \varphi^{o \rightarrow o} x^{o}. \varphi \ x )(f \ x)
   \]

   leads to variable capture:

   \[
   (\lambda \varphi x. \varphi x)(f \ x) \not\rightarrow_{\beta} (\lambda x. (f \ x) x).
   \]

   Hence the term is unsafe. Indeed, $\text{ord} \ x = 0 \leq 1 = \text{ord} f \ x$.

2. The term $(\lambda \varphi^{o \rightarrow o} x^{o}. \varphi \ x)(\lambda y^{o}. y)$ is safe.

3. Safety does not capture “variable-renaming uselessness”. E.g. the unsafe term $\lambda y^{o} z^{o}. (\lambda x^{o}. y) z$ can be contracted using capture-permitting substitution.

4. Up to order 2, $\beta$-normal terms are always safe.

5. Kierstead terms $\lambda f((o, o), o). f(\lambda x^{o}. f(\lambda y^{o}. y))$ is safe but $\lambda f((o, o), o). f(\lambda x^{o}. f(\lambda y^{o}. x))$ is unsafe.
Examples

1. Contracting the $\beta$-redex in the following term

$$f : o \to o \to o, \ x : o \vdash (\lambda \varphi^{o \to o} x^o. \varphi \ x)(\underline{f \ x})$$

leads to variable capture:

$$(\lambda \varphi x. \varphi \ x)(f \ x) \not\to_{\beta} (\lambda x. (f \ x) x).$$

Hence the term is unsafe. Indeed, $\text{ord} \ x = 0 \leq 1 = \text{ord} f \ x$.

2. The term $$(\lambda \varphi^{o \to o} x^o. \varphi \ x)(\lambda y^o. y)$$ is safe.

3. Safety does not capture “variable-renaming uselessness”. E.g. the unsafe term $\lambda y^o z^o. (\lambda x^o. y) z$ can be contracted using capture-permitting substitution.

4. Up to order 2, $\beta$-normal terms are always safe.

5. Kierstead terms $\lambda f((o,o),o). f(\lambda x^o. f(\lambda y^o. y))$ is safe but $\lambda f((o,o),o). f(\lambda x^o. f(\underline{\lambda y^o}. x))$ is unsafe.
Examples

1. Contracting the $\beta$-redex in the following term

$$f : o \rightarrow o \rightarrow o, x : o \vdash (\lambda \varphi^{o \rightarrow o} x^{o} . \varphi \ x)(\underline{x})$$

leads to variable capture:

$$(\lambda \varphi x . \varphi \ x)(f \ x) \not\beta (\lambda x . (f \ x)x).$$

Hence the term is unsafe. Indeed, $\text{ord} \ x = 0 \leq 1 = \text{ord} f \ x$.

2. The term $$(\lambda \varphi^{o \rightarrow o} x^{o} . \varphi \ x)(\lambda y^{o} . y)$$ is safe.

3. Safety does not capture “variable-renaming uselessness”. E.g. the unsafe term $\lambda y^{o} z^{o} . (\lambda x^{o} . y) z$ can be contracted using capture-permitting substitution.

4. Up to order 2, $\beta$-normal terms are always safe.

5. Kierstead terms $\lambda f^{((o, o), o)} . f(\lambda x^{o} . f(\lambda y^{o} . y))$ is safe but $\lambda f^{((o, o), o)} . f(\lambda x^{o} . f(\lambda y^{o} . x))$ is unsafe.
Examples

1. Contracting the $\beta$-redex in the following term 

   \[ f : o \rightarrow o \rightarrow o, x : o \vdash (\lambda o \rightarrow o x^o. o x)(x) \]

   leads to variable capture:

   \[(\lambda o x. o x)(x) \not \rightarrow_{\beta} (\lambda x. (f x)x).\]

   Hence the term is unsafe. Indeed, $\text{ord } x = 0 \leq 1 = \text{ord } f x$.

2. The term $(\lambda o \rightarrow o x^o. o x)(\lambda y^o. y)$ is safe.

3. Safety does not capture “variable-renaming uselessness". E.g. the unsafe term $\lambda y^o z^o. (\lambda x^o. y)z$ can be contracted using capture-permitting substitution.

4. Up to order 2, $\beta$-normal terms are always safe.

5. Kierstead terms $\lambda f ((o,o),o). f (\lambda x^o. f (\lambda y^o. y))$ is safe but $\lambda f ((o,o),o). f (\lambda x^o. f (\lambda y^o. x))$ is unsafe.
Examples

1. Contracting the $\beta$-redex in the following term

   $$f : o \rightarrow o \rightarrow o, x : o \vdash (\lambda \varphi^{o \rightarrow o^{o}} \cdot \varphi^{o} x^{x})(f^{x})$$

   leads to variable capture:

   $$(\lambda \varphi x. \varphi^{o} x^{x})(f^{x}) \not\rightarrow^{\beta} (\lambda x. (f^{x} x^{x}))^{x}.$$ 

   Hence the term is unsafe. Indeed, $\text{ord } x = 0 \leq 1 = \text{ord } f^{x}$.

2. The term $(\lambda \varphi^{o \rightarrow o^{o}} \cdot \varphi^{o} x^{x})(\lambda y^{o} y^{y})$ is safe.

3. Safety does not capture “variable-renaming uselessness”. E.g. the unsafe term $\lambda y^{o} z^{o}. (\lambda x^{o} y^{y}) z^{z}$ can be contracted using capture-permitting substitution.

4. Up to order 2, $\beta$-normal terms are always safe.

5. Kierstead terms $\lambda f^{((o, o)^{o}, o^{o})^{o}}. f^{(\lambda x^{o} f^{(\lambda y^{o} y^{y})})}$ is safe but $\lambda f^{((o, o)^{o}, o^{o})^{o}}. f^{(\lambda x^{o} f^{(\lambda y^{o} x^{x})})}$ is unsafe.
Examples

1. Contracting the $\beta$-redex in the following term

$$f : o \rightarrow o \rightarrow o, \ x : o \vdash (\lambda \varphi \cdot \varphi \ x)(f \ x)$$

leads to variable capture:

$$(\lambda \varphi x. \varphi x)(f \ x) \not\rightarrow_\beta (\lambda x.(f \ x)x).$$

Hence the term is unsafe. Indeed, $\text{ord } x = 0 \leq 1 = \text{ord } f \ x$.

2. The term $(\lambda \varphi \cdot \varphi \ x)(\lambda y. y)$ is safe.

3. Safety does not capture “variable-renaming uselessness”. E.g. the unsafe term $\lambda y \cdot z. (\lambda x \cdot y)z$ can be contracted using capture-permitting substitution.

4. Up to order 2, $\beta$-normal terms are always safe.

5. Kierstead terms $\lambda f^{(\circ,\circ)} \cdot f(\lambda x \cdot f(\lambda y \cdot y))$ is safe but $\lambda f^{(\circ,\circ)} \cdot f(\lambda x \cdot f(\lambda y \cdot x))$ is unsafe.
Transformations preserving safety

- Substitution preserves safety.
- $\beta$-reduction does not preserve safety: Take $w, x, y, z : o$ and $f : (o, o, o)$. The safe term $(\lambda xy. f x y)z w$ $\beta$-reduces to the unsafe term $(\lambda y. f z y)w$ which in turns reduces to the safe term $f z w$.
- Safe $\beta$-reduction: reduces simultaneously as many $\beta$-redexes as needed in order to reach a safe term.
- Safe $\beta$-reduction preserves safety.
- $\eta$-reduction preserves safety.
- $\eta$-expansion does not preserve safety.
  E.g. $\vdash_s \lambda y^o z^o y : (o, o, o)$ but $\not\vdash_s \lambda x^o. (\lambda y^o z^o y) x : (o, o, o)$.
- $\eta$-long normal expansion preserves safety.
Transformations preserving safety

- Substitution preserves safety.

- $\beta$-reduction does not preserve safety: Take $w, x, y, z : o$ and $f : (o, o, o)$. The safe term $(\lambda xy.f \times y)z\ w$ $\beta$-reduces to the unsafe term $(\lambda y.f\ z\ y)w$ which in turns reduces to the safe term $f\ z\ w$.

- Safe $\beta$-reduction: reduces simultaneously as many $\beta$-redexes as needed in order to reach a safe term.

- Safe $\beta$-reduction preserves safety.

- $\eta$-reduction preserves safety.

- $\eta$-expansion does not preserve safety.
  E.g. $\vdash_s \lambda y^o z^o. y : (o, o, o)$ but $\not\vdash_s \lambda x^o.(\lambda y^o z^o. y)x : (o, o, o)$.

- $\eta$-long normal expansion preserves safety.
Transformations preserving safety

- Substitution preserves safety.
- $\beta$-reduction does not preserve safety: Take $w, x, y, z : o$ and $f : (o, o, o)$. The safe term $(\lambda xy.f \times y)z \ w$ $\beta$-reduces to the unsafe term $(\lambda y.f \ z \ y)w$ which in turns reduces to the safe term $f \ z \ w$.
- Safe $\beta$-reduction: reduces simultaneously as many $\beta$-redexes as needed in order to reach a safe term.
- Safe $\beta$-reduction preserves safety.
- $\eta$-reduction preserves safety.
- $\eta$-expansion does not preserve safety. E.g. $\vdash_s \lambda y^o z^o.y : (o, o, o)$ but $\not\vdash_s \lambda x^o.(\lambda y^o z^o.y)x : (o, o, o)$.
- $\eta$-long normal expansion preserves safety.
Transformations preserving safety

- Substitution preserves safety.
- \( \beta \)-reduction does not preserve safety: Take \( w, x, y, z : o \) and \( f : (o, o, o) \). The safe term \((\lambda xy.f \times y)z w\) \( \beta \)-reduces to the unsafe term \((\lambda y.f z y)w\) which in turns reduces to the safe term \( f z w \).
- Safe \( \beta \)-reduction: reduces simultaneously as many \( \beta \)-redexes as needed in order to reach a safe term.
- Safe \( \beta \)-reduction preserves safety.
- \( \eta \)-reduction preserves safety.
- \( \eta \)-expansion does not preserve safety. E.g. \( \vdash_s \lambda y^o z^o . y : (o, o, o) \) but \( \not\vdash_s \lambda x^o . (\lambda y^o z^o . y)x : (o, o, o) \).
- \( \eta \)-long normal expansion preserves safety.
Transformations preserving safety

- Substitution preserves safety.
- $\beta$-reduction does not preserve safety: Take $w, x, y, z : o$ and $f : (o, o, o)$. The safe term $(\lambda xy.f \times y)z w$ $\beta$-reduces to the unsafe term $(\lambda y.f z y)w$ which in turns reduces to the safe term $f z w$.
- Safe $\beta$-reduction: reduces simultaneously as many $\beta$-redexes as needed in order to reach a safe term.
- Safe $\beta$-reduction preserves safety.
- $\eta$-reduction preserves safety.
- $\eta$-expansion does not preserve safety. E.g. $\vdash_s \lambda y^o z^o . y : (o, o, o)$ but $\not\vdash_s \lambda x^o . (\lambda y^o z^o . y) x : (o, o, o)$.
- $\eta$-long normal expansion preserves safety.
Transformations preserving safety

- Substitution preserves safety.
- $\beta$-reduction does not preserve safety: Take $w, x, y, z : o$ and $f : (o, o, o)$. The safe term $(\lambda xy.f \times y)z w$ $\beta$-reduces to the unsafe term $(\lambda y.f \ z \ y)w$ which in turns reduces to the safe term $f \ z \ w$.
- Safe $\beta$-reduction: reduces simultaneously as many $\beta$-redexes as needed in order to reach a safe term.
- Safe $\beta$-reduction preserves safety.
- $\eta$-reduction preserves safety.
- $\eta$-expansion does not preserve safety.
  E.g. $\vdash_s \lambda y^o z^o. y : (o, o, o)$ but $\not\vdash_s \lambda x^o.(\lambda y^o z^o. y)x : (o, o, o)$.
- $\eta$-long normal expansion preserves safety.
Expressivity

Safety is a strong constraint but it is still unclear how it restricts expressivity:

- de Miranda showed that at order 2 for word languages, non-determinism palliates the loss of expressivity. It is unknown if this extends to higher orders.
- For tree-generating grammars: Urzyczyn conjectured that safety is a proper constraint i.e. that there is a tree which is intrinsically unsafe. He proposed a possible counter-example.
- For graphs, HMOS06’s undecidability result implies that safety restricts expressivity.
- For simply-typed terms: ...
Numerical functions

Church Encoding: for \( n \in \mathbb{N} \), \( \overline{n} = \lambda sz.s^n z \) of type \( I = (o \to o) \to o \to o \).

Theorem (Schwichtenberg 1976)

The numeric functions representable by simply-typed terms of type \( I \to \ldots \to I \) are exactly the multivariate polynomials extended with the conditional function:

\[
\text{cond}(t, x, y) = \begin{cases} 
  x, & \text{if } t = 0 \\
  y, & \text{if } t = n + 1 \end{cases}.
\]

\( \text{cond} \) is represented by the term \( C = \lambda FGH\alpha x.H(\lambda y.G\alpha x)(F\alpha x) \).

Theorem

Functions representable by safe \( \lambda \)-expressions of type \( I \to \ldots \to I \) are exactly the multivariate polynomials.

So \( \text{cond} \) is not representable in the Safe \( \lambda \)-calculus and \( C \) is unsafe.
Numerical functions

Church Encoding: for \( n \in \mathbb{N}, \overline{n} = \lambda sz.s^n z \) of type 
\( I = (o \to o) \to o \to o. \)

**Theorem (Schwichtenberg 1976)**

The numeric functions representable by simply-typed terms of type
\( I \to \ldots \to I \) are exactly the multivariate polynomials extended with
the conditional function:

\[
\text{cond}(t, x, y) = \begin{cases} 
  x, & \text{if } t = 0 \\
  y, & \text{if } t = n + 1
\end{cases}.
\]

\( \text{cond} \) is represented by the term 
\( C = \lambda FGH\alpha x. H(\lambda y. G\alpha x)(F\alpha x). \)

**Theorem**

Functions representable by safe \( \lambda \)-expressions of type \( I \to \ldots \to I \)
are exactly the multivariate polynomials.

So \( \text{cond} \) is not representable in the Safe \( \lambda \)-calculus and \( C \) is unsafe.
Part II : Game semantics
Game semantics

Model of programming languages based on games (Abramsky et al.; Hyland and Ong; Nickau)

- 2 players: Opponent (system) and Proponent (program)
- The term type induces an arena defining the possible moves

\[
\begin{align*}
\mathbb{N} &= q_0 \quad \mathbb{N} \rightarrow \mathbb{N} = q^0 \\
0 &\quad 1 & \quad \cdots \\
0 &\quad 1 & \quad \cdots
\end{align*}
\]

- Play = sequence of moves played alternatively by O and P with justification pointers.

- Strategy for P = prefix-closed set of plays. \( sab \) in the strategy means that P should respond \( b \) when O plays \( a \) in position \( s \).

- The denotation of a term \( M \), written \( \llbracket M \rrbracket \), is a strategy for P.

\[
\begin{align*}
\llbracket 7 : \mathbb{N} \rrbracket &= \{ \epsilon, q, q \ 7 \} \\
\llbracket \text{succ} : \mathbb{N} \rightarrow \mathbb{N} \rrbracket &= \text{Pref}(\{ q^0 q^1 n(n + 1) \mid n \in \mathbb{N} \})
\end{align*}
\]

- Compositionality: \( \llbracket \text{succ} \ 7 \rrbracket = \llbracket \text{succ} \rrbracket ; \llbracket 7 \rrbracket \)
Computation trees and traversals

*Computation tree:* AST of the $\eta$-long normal form of a term.
Example: $M \equiv \lambda f z. (\lambda g x. f x)(\lambda y. y) z$ of type $(o \rightarrow o) \rightarrow o \rightarrow o$. 

```
λfz
  @
λgx    λy
  f     y
  λ     z
  x
```
Computation trees and traversals

*Computation tree:* AST of the $\eta$-long normal form of a term.
Example: $M \equiv \lambda fz. (\lambda gx.fx)(\lambda y.y)z$ of type $(o \to o) \to o \to o$.

*Traversal:* justified sequence of nodes representing the computation.
Computation trees and traversals

**Computation tree:** AST of the $\eta$-long normal form of a term.
Example: $M \equiv \lambda fz.(\lambda gx.fx)(\lambda y.y)z$ of type $(o \to o) \to o \to o$.

**Traversal:** justified sequence of nodes representing the computation.

$$t = \lambda fz$$

```
    λfz
     @
   /  \
λgx \ λy \ λ
   |  /  /
  f  y  z
  /  /
λ λ
/  /
x x
```
Computation trees and traversals

**Computation tree:** AST of the \( \eta \)-long normal form of a term. Example: \( M \equiv \lambda fz.(\lambda gx.fx)(\lambda y.y)z \) of type \( (o \to o) \to o \to o \).

**Traversal:** justified sequence of nodes representing the computation.

\[
t = \lambda fz \cdot @
\]
Computation trees and traversals

**Computation tree:** AST of the $\eta$-long normal form of a term.
Example: $M \equiv \lambda fz.(\lambda gx.fx)(\lambda y.y)z$ of type $(o \to o) \to o \to o$.

**Traversal:** justified sequence of nodes representing the computation.

$t = \lambda fz \cdot @ \cdot \lambda gx$
Computation trees and traversals

*Computation tree:* AST of the $\eta$-long normal form of a term.
Example: $M \equiv \lambda fz. (\lambda gx.fx)(\lambda y.y)z$ of type $(o \to o) \to o \to o$.

*Traversal:* justified sequence of nodes representing the computation.

$$t = \lambda fz \cdot @ \cdot \lambda gx \cdot f$$
Computation trees and traversals

**Computation tree:** AST of the \( \eta \)-long normal form of a term.
Example: \( M \equiv \lambda f z. (\lambda g x. f x)(\lambda y. y) z \) of type \( (o \to o) \to o \to o \).

**Traversal:** justified sequence of nodes representing the computation.

\[
t = \lambda f z \cdot @ \cdot \lambda g x \cdot f \cdot \lambda
\]
Computation trees and traversals

*Computation tree:* AST of the $\eta$-long normal form of a term.
Example: $M \equiv \lambda f z. (\lambda g x. f x)(\lambda y. y) z$ of type $(o \to o) \to o \to o$.

*Traversal:* justified sequence of nodes representing the computation.

$$t = \lambda f z \cdot @ \cdot \lambda g x \cdot f \cdot \lambda \cdot x$$
Computation trees and traversals

**Computation tree:** AST of the $\eta$-long normal form of a term.
Example: $M \equiv \lambda fz. (\lambda gx.fx)(\lambda y.y)z$ of type $(o \rightarrow o) \rightarrow o \rightarrow o$.

**Traversal:** justified sequence of nodes representing the computation.

\[
t = \lambda fz \cdot @ \cdot \lambda gx \cdot f \cdot \lambda \cdot x \cdot \lambda
\]
Computation trees and traversals

**Computation tree:** AST of the \( \eta \)-long normal form of a term. Example: \( M \equiv \lambda fz.(\lambda gx.fx)(\lambda y.y)z \) of type \( (o \rightarrow o) \rightarrow o \rightarrow o \).

**Traversal:** justified sequence of nodes representing the computation.

\[ t = \lambda fz \cdot @ \cdot \lambda gx \cdot f \cdot \lambda \cdot x \cdot \lambda \cdot z \]
Computation trees and traversals

**Computation tree:** AST of the $\eta$-long normal form of a term.
Example: $M \equiv \lambda f z. (\lambda g x . f x) (\lambda y . y) z$ of type $(o \to o) \to o \to o$.

**Traversal:** justified sequence of nodes representing the computation.

\[
t = \lambda f z \cdot @ \cdot \lambda g x \cdot f \cdot \lambda \cdot x \cdot \lambda \cdot z
\]

**Traversal reduction:** keep only nodes hereditarily justified by the root.

\[
t \upharpoonright r = \lambda f z \cdot f \cdot \lambda \cdot z
\]
Computation trees and traversals

Computation tree: AST of the \( \eta \)-long normal form of a term.
Example: \( M \equiv \lambda fz. (\lambda gx.fx)(\lambda y.y)z \) of type \( (o \to o) \to o \to o \).

Traversal: justified sequence of nodes representing the computation.

\[
t = \lambda fz \cdot \emptyset \cdot \lambda gx \cdot f \cdot \lambda \cdot x \cdot \lambda \cdot z
\]

Traversal reduction: keep only nodes hereditarily justified by the root.

\[
t \upharpoonright r = \lambda fz \cdot f \cdot \lambda \cdot z
\]

\( \emptyset \)-nodes removal:

\[
t - \emptyset = \lambda fz \cdot \lambda gx \cdot f \cdot \lambda \cdot x \cdot \lambda \cdot z
\]
The Correspondence Theorem

Let $M$ be a simply typed term of type $T$. There exists a partial function $\varphi$ from the nodes of the computation tree to the moves of the arena $[T]$ such that

$$\varphi : Trav(M) \Downarrow \circ \xrightarrow{ir} \langle \langle M \rangle \rangle$$

$$\varphi : Trav(M) \uparrow r \xrightarrow{ir} [M] .$$

where

- $\text{Trav}(M)$ = set of traversals of the computation tree of $M$
- $\text{Trav}(M) \uparrow r = \{ t \uparrow r \mid t \in \text{Trav}(M) \}$
- $\text{Trav}(M) \Downarrow \circ = \{ t \Downarrow \circ \mid t \in \text{Trav}(M) \}$
- $[M]$ = game-semantic denotation of $M$
- $\langle \langle M \rangle \rangle$ = revealed denotation (i.e. internal moves are uncovered.)
The Correspondence Theorem (example)

Left: computation tree. Right: arena.

Take the traversal \( t = \lambda f z \cdot \emptyset \cdot \lambda g x \cdot f \cdot \lambda \cdot x \cdot \lambda \cdot z \). We have:

\[ \varphi(t \upharpoonright r) = \varphi(\lambda f z \cdot f \cdot \lambda \cdot z) = q^1 q^3 q^4 q^2 \in [M]. \]
The Correspondence Theorem (2)

<table>
<thead>
<tr>
<th>Computation tree notions</th>
<th>Game-semantic equivalents</th>
</tr>
</thead>
<tbody>
<tr>
<td>computation tree</td>
<td>arena(s)</td>
</tr>
<tr>
<td>traversal</td>
<td>uncovered play</td>
</tr>
<tr>
<td>reduced traversal</td>
<td>play</td>
</tr>
<tr>
<td>path in the computation tree</td>
<td>P-view of an uncovered play</td>
</tr>
</tbody>
</table>
Game-semantic Characterisation of Safety

- The computation tree of a safe term is **incrementally-bound**: each variable $x$ is bound by the first $\lambda$-node occurring in the path to the root with order $\text{ord } x > \text{ord } x$.

- By the Correspondence Theorem, this implies that:

**Proposition**

- Safe terms are denoted by **P-incrementally justified strategies**: each P-move $m$ points to the last O-move in the P-view with order $\text{ord } m$.

- Reciprocally, if a *closed* term is denoted by a P-incrementally justified strategy then its $\eta$-long $\beta$-normal form is safe.

**Corollary**

Justification pointers attached to P-moves are redundant in the game-semantics of safe terms.
Game-semantic Characterisation of Safety

- The computation tree of a safe term is incrementally-bound: each variable $x$ is bound by the first $\lambda$-node occurring in the path to the root with order $> \text{ord } x$.
- By the Correspondence Theorem, this implies that:

**Proposition**

- Safe terms are denoted by $P$-incrementally justified strategies: each $P$-move $m$ points to the last $O$-move in the $P$-view with order $> \text{ord } m$.
- Reciprocally, if a closed term is denoted by a $P$-incrementally justified strategy then its $\eta$-long $\beta$-normal form is safe.

**Corollary**

Justification pointers attached to $P$-moves are redundant in the game-semantics of safe terms.
Game-semantic Characterisation of Safety

- The computation tree of a safe term is incrementally-bound: each variable $x$ is bound by the first $\lambda$-node occurring in the path to the root with order $> \text{ord } x$.

- By the Correspondence Theorem, this implies that:

  **Proposition**

  - Safe terms are denoted by P-incrementally justified strategies: each P-move $m$ points to the last O-move in the P-view with order $> \text{ord } m$.

  - Reciprocally, if a closed term is denoted by a P-incrementally justified strategy then its $\eta$-long $\beta$-normal form is safe.

  **Corollary**

  Justification pointers attached to P-moves are redundant in the game-semantics of safe terms.
Question Do P-incrementally-justified strategies compose?
No. Take \( \sigma = \llbracket \vdash_s \lambda x^o v^o . x : o \to (o, o) \rrbracket \) and
\( \mu = \llbracket \vdash_s \lambda y^{(o, o)} \varphi((o,o),o) . \varphi(\lambda u^o . ya) : (o, o) \to (((o, o), o), o) \rrbracket \) for
some constant \( a : o \). We have \( \sigma \circ \mu = \llbracket \lambda x \varphi . \varphi(\lambda u . x) \rrbracket \) which is not
P-i.j. by the previous proposition.
Definition

A strategy $\sigma : A \rightarrow B$ is **closed P-incrementally justified** if it P-i.j. and if for every move $m$ initial in $A$ that is contained in some play of $\sigma$ we have $\text{ord}_A m \geq \text{ord } B$.

- **Remark:** This property is not preserved up to the Curry isomorphism!
- **Example:** any P-i.j. strategy on $I \rightarrow A$ is closed P-i.j.
- **Example:** Safe terms denotations are closed P-i.j.

**Proposition**

Closed P-incrementally justified strategies compose.

Hence we have:

- a category of games and closed P-i.j. strategies,
- that is not cartesian-closed,
- which models the safe $\lambda$-calculus.
Compositionality 2

Definition

A strategy $\sigma : A \to B$ is closed P-incrementally justified if it P-i.j. and if for every move $m$ initial in $A$ that is contained in some play of $\sigma$ we have $\text{ord}_A m \geq \text{ord} B$.

Remark: This property is not preserved up to the Curry isomorphism!

Example: any P-i.j. strategy on $I \to A$ is closed P-i.j.

Safe terms denotations are closed P-i.j.

Proposition

Closed P-incrementally justified strategies compose.

Hence we have:

- a category of games and closed P-i.j. strategies,
- that is not cartesian-closed,
- which models the safe $\lambda$-calculus.
Safe PCF

- \( \text{PCF} = \lambda \to \) with base type \( \mathbb{N} + \) successor, predecessor, conditional + Y combinator
- \( \text{Safe PCF} = \) Safe fragment of PCF

Proposition

Safe PCF terms are denoted by closed P-i.j. strategies.

Definability

Let \( \sigma \) be a well-bracketed innocent P-i.j. strategy with finite view function defined on a PCF arena \( A_1 \times \ldots \times A_i \to B \). \( \sigma \) is the denotation of some term \( \overline{x} : \overline{A} \vdash M : B \) such that \( \lambda \overline{x}. M \) is safe.

Question: Does this give a fully abstract model with respect to safe contexts? Problem: The quotiented category model is not rational (since it is not even cartesian closed)!
Conclusion and Future Works

**Conclusion:**
Safety is a syntactic constraint with interesting algorithmic and game-semantic properties.

**Future works:**
- Is there a fully abstract model of Safe PCF (with respect to safe contexts)?
- Complexity classes characterised with the Safe $\lambda$-calculus?
- Safe Idealized Algol: is contextual equivalence decidable for some finitary fragment (e.g. Safe IA$_4$) (with respect to all/safe contexts)?

**Related works:**
- Jolie G. de Miranda’s thesis on safe/unsafe grammars.
- Ong introduced computation trees in LICS2006 to prove decidability of MSO theory on infinite trees generated by higher-order grammars (whether safe or not).