The Safe λ-Calculus


Oxford University Computing Laboratory

TLCA 2007
Overview

- Safety is originally a syntactic restriction for higher-order grammars with nice automata-theoretic characterization.
- In the context of the λ-calculus it gives rise to the Safe λ-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

1. The safety restriction for higher-order grammars
2. The safe $\lambda$-calculus
3. Expressivity
4. Game-semantic characterization
5. Safe PCF, Safe IA
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 are used as generators of word languages (Maslov, 1974), trees (KNU01) or graphs.
- A higher-order grammar is formally given by a tuple $\langle \Sigma, N, R, S \rangle$ (terminals, non-terminals, rewriting rules, starting symbol)
- Example of a tree-generating order-2 grammar:

$$
S \rightarrow H \ a \\
H \ z^o \rightarrow F (g \ z) \\
F \ \phi^{(o,o)} \rightarrow \phi (\phi (F \ h)) \\
$$

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

Diagram:

```
         g
        / \  \
        a   h
       /    \
      b     h
      /     \
     a     h
    /      \
   b       h
   /       \
  a         h
```

Non-terminals: $S : o$, $H : (o, o)$ and $F : ((o, o), o)$.
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.
  
  - \((A_1, \cdots, A_n, o)\) 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.
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.
Safe grammars: examples

Take \( h : o \rightarrow o, \ g : o \rightarrow o \rightarrow o, \ a : o \).
The following grammar is unsafe:

\[
\begin{align*}
S & \rightarrow H a \\
H z^o & \rightarrow F (g z) \\
F \phi^{(o,o)} & \rightarrow \phi (\phi (F h))
\end{align*}
\]

It is equivalent to the following safe grammar:

\[
\begin{align*}
S & \rightarrow F(g a) \\
F \phi^{(o,o)} & \rightarrow \phi (\phi (F h))
\end{align*}
\]
Safe grammars: examples

Take \( h : o \rightarrow o, \ g : o \rightarrow o \rightarrow o, \ a : o. \)
The following grammar is unsafe:

\[
\begin{align*}
S & \rightarrow H a \\
H z^o & \rightarrow F (g z) \\
F \phi^{(o,o)} & \rightarrow \phi (\phi (F h))
\end{align*}
\]

It is equivalent to the following safe grammar:

\[
\begin{align*}
S & \rightarrow F(g a) \\
F \phi^{(o,o)} & \rightarrow \phi (\phi (F h))
\end{align*}
\]
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 Cauca’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** $A := o \mid A \rightarrow A$.

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

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

  \[
  \begin{align*}
  (\text{var}) & \quad \frac{}{x : A \vdash x : A} \\
  (\text{wk}) & \quad \frac{\Gamma \vdash M : A}{\Delta \vdash M : A} \quad \Gamma \subset \Delta \\
  (\text{app}) & \quad \frac{\Gamma \vdash M : A \rightarrow B \quad \Gamma \vdash N : A}{\Gamma \vdash MN : B} \\
  (\text{abs}) & \quad \frac{\Gamma, x : A \vdash M : B}{\Gamma \vdash \lambda x^A. M : A \rightarrow B}
  \end{align*}
  \]

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

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

The formation rules

$\textbf{(var)} \quad x : A \vdash_s x : A$

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

$\textbf{(app)} \quad \frac{\Gamma \vdash M : (A_1, \ldots, A_l, B) \quad \Gamma \vdash_s N_1 : A_1 \quad \ldots \quad \Gamma \vdash_s N_l : A_l}{\Gamma \vdash_s MN_1 \ldots N_l : B}$

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

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

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 $\text{ord } A$.***
The Safe $\lambda$-Calculus: examples

- Some examples of safe terms: $\lambda x.x$, $\lambda x y.x$, $\lambda x y.y$.
- Up to order 2, $\beta$-normal terms are always safe.
- The two Kierstead terms (order 3). Only one of them is safe:
  \[
  \lambda f ((o, o), o) \cdot f (\lambda x^o.f (\lambda y^o.y)) \\
  \lambda f ((o, o), o) \cdot f (\lambda x^o.f (\lambda y^o.x))
  \]
- An example of safe term not in $\beta$-normal form:
  \[
  (\lambda \varphi^o \mapsto o x^o.\varphi x)(\lambda y^o.y)
  \]
The Safe $\lambda$-Calculus: examples

- Some examples of safe terms: $\lambda x.x$, $\lambda xy.x$, $\lambda xy.y$.
- Up to order 2, $\beta$-normal terms are always safe.
- The two Kierstead terms (order 3). Only one of them is safe:

  $\lambda f((o,o), o).f(\lambda x^o.f(\lambda y^o.y))$

  $\lambda f((o,o), o).f(\lambda x^o.f(\lambda y^o.x))$

- An example of safe term not in $\beta$-normal form:

  $(\lambda \varphi^o \to o x^o.\varphi x)(\lambda y^o.y)$
Some examples of safe terms: $\lambda x.x$, $\lambda xy.x$, $\lambda xy.y$.

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

The two Kierstead terms (order 3). Only one of them is safe:

$$\lambda f((o,o),o).f(\lambda x^o.f(\lambda y^o.y))$$
$$\lambda f((o,o),o).f(\lambda x^o.f(\lambda y^o.x))$$

An example of safe term not in $\beta$-normal form:

$$(\lambda \varphi^\rightarrow^o x^o.\varphi \ x)(\lambda y^o.y)$$
Some examples of safe terms: $\lambda x.x$, $\lambda xy.x$, $\lambda xy.y$.

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

The two Kierstead terms (order 3). Only one of them is safe:

$$\lambda f((x,x),x).f((x^{\circ}.f(x^{\circ}.x)))$$
$$\lambda f((x,x),x).f((x^{\circ}.f(x^{\circ}.x)))$$

An example of safe term not in $\beta$-normal form:

$$(\lambda f^\circ.\phi^\circ x^\circ.\phi x)(\lambda y^{\circ}.y)$$
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 β-reductions.

2. **Another solution**: use the λ-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 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. $(\lambda x.(\lambda y.x))y$ 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: examples

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

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

leads to variable capture:

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

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

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

3. The unsafe term $\lambda y^o z^o.(\lambda x^o.y)z$ can be contracted without renaming variables. Hence not all terms whose $\beta$-contraction can be correctly implemented by capture permitting substitution, are safe.
Variable capture: examples

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

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

   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 \((\lambda \varphi o \rightarrow o \cdot \varphi \ x)(\lambda y o . y)\) is safe.

3. The unsafe term \(\lambda y o z o . (\lambda x o . y)z\) can be contracted without renaming variables. Hence not all terms whose $\beta$-contraction can be correctly implemented by capture permitting substitution, are safe.
Variable capture: examples

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

\[ f : o \to o \to o, \ x : o \vdash (\lambda o \to o \cdot o \cdot \varphi \ x)(f \ x) \]

leads to variable capture:

\[ (\lambda o \cdot o \cdot \varphi \ x)(f \ x) \not\to \beta (\lambda \cdot o \cdot (f \ x) \cdot o \cdot x) \cdot o \cdot x.\]

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

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

3. The unsafe term $\lambda o \cdot o \cdot z \cdot z \cdot (\lambda o \cdot o \cdot y \cdot y) z$ can be contracted without renaming variables. Hence not all terms whose $\beta$-contraction can be correctly implemented by capture permitting substitution, are safe.
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:

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

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. The unsafe term $\lambda y^o z^o.(\lambda x^o.y)z$ can be contracted without renaming variables. Hence not all terms whose $\beta$-contraction can be correctly implemented by capture permitting substitution, are safe.
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. $\vdash \lambda y^o z^o.y : (o, o, o)$ but $\nvdash \lambda x^o.((\lambda y^o z^o.y)x : (o, o, o))$.
- $\eta$-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 and Ong 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: ...
Church Encoding: for $n \in \mathbb{N}$, $\bar{n} = \lambda sz. s^n z$ 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:

$$cond(t, x, y) = \begin{cases} x, & \text{if } t = 0 \\ y, & \text{if } t = n + 1 \end{cases}.$$
Numerical functions (2)

Let $n, m \in \mathbb{N}$.

- **Natural number:** $\overline{n} = \lambda sz.s^nz : (o \rightarrow o) \rightarrow o \rightarrow o$. Safe.
- **Addition:** $\overline{n + m} = \lambda x.\alpha^0(x^0)(\overline{n} \alpha)(\overline{m} \alpha x)$. Safe.
- **Multiplication:** $\overline{n \cdot m} = \lambda x.\alpha^0(\overline{n} \alpha)(\overline{m} \alpha)$. Safe.
- **Conditional:** $C = \lambda FGH \alpha x.H(\lambda y.G \alpha x)(F \alpha x)$. Unsafe.

In fact:

**Theorem**

*Functions representable by safe $\lambda$-expressions of type $I \rightarrow \ldots \rightarrow I$ are exactly the multivariate polynomials.*
Numerical functions (2)

Let $n, m \in \mathbb{N}$.

- **Natural number:** $\bar{n} = \lambda sz.s^n z : (o \to o) \to o \to o$. Safe.
- **Addition:** $\bar{n} + \bar{m} = \lambda \alpha^{(o, o)}x^o.(\bar{n} \alpha)(\bar{m} \alpha x)$. Safe.
- **Multiplication:** $\bar{n}.\bar{m} = \lambda \alpha^{(o, o)}\bar{n}(\bar{m} \alpha)$. Safe.
- **Conditional:** $C = \lambda FGH\alpha x.H(\lambda y.G\alpha x)(F\alpha x)$. Unsafe.

In fact:

**Theorem**

*Functions representable by safe $\lambda$-expressions of type $I \to \cdots \to I$ are exactly the multivariate polynomials.*
Let \( n, m \in \mathbb{N} \).

- **Natural number:** \( \bar{n} = \lambda sz. s^n z : (o \to o) \to o \to o \). Safe.
- **Addition:** \( \bar{n} + \bar{m} = \lambda \alpha^{(o, o)} x^o. (\bar{n} \alpha)(\bar{m} \alpha x) \). Safe.
- **Multiplication:** \( \bar{n} \cdot \bar{m} = \lambda \alpha^{(o, o)} \bar{n} (\bar{m} \alpha) \). Safe.
- **Conditional:** \( C = \lambda FGH \alpha x. H(\underline{\lambda y. G} \alpha x)(F \alpha x) \). Unsafe.

In fact:

**Theorem**

*Functions representable by safe \( \lambda \)-expressions of type \( I \to \ldots \to I \) are exactly the multivariate polynomials.*
Numerical functions (2)

Let \( n, m \in \mathbb{N} \).

- **Natural number:** \( \bar{n} = \lambda sz. s^n z : (o \to o) \to o \to o \). Safe.
- **Addition:** \( \bar{n} + \bar{m} = \lambda \alpha^{(o, o)} x^o. (\bar{n} \alpha)(\bar{m} \alpha x) \). Safe.
- **Multiplication:** \( \bar{n} \cdot \bar{m} = \lambda \alpha^{(o, o)} . \bar{n}(\bar{m} \alpha) \). Safe.
- **Conditional:** \( C = \lambda FGH\alpha x. H(\lambda y. G\alpha x)(F\alpha x) \). Unsafe.

In fact:

**Theorem**

*Functions representable by safe \( \lambda \)-expressions of type \( I \to \ldots \to I \) are exactly the multivariate polynomials.*
Numerical functions (2)

Let \( n, m \in \mathbb{N} \).

- Natural number: \( \overline{n} = \lambda sz.s^n z : (o \rightarrow o) \rightarrow o \rightarrow o \). Safe.
- Addition: \( \overline{n + m} = \lambda \alpha^{(o,o)} x^o. (\overline{n} \alpha)(\overline{m} \alpha x) \). Safe.
- Multiplication: \( \overline{n \cdot m} = \lambda \alpha^{(o \cdot o)} . \overline{n} (\overline{m} \alpha) \). Safe.
- Conditional: \( C = \lambda FGH \alpha x . H(\lambda y . G \alpha x)(F \alpha x) \). Unsafe.

In fact:

**Theorem**

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

\[
\begin{align*}
\llbracket N \rrbracket &= q \quad \begin{array}{c}
q_0 \\
0 \\
1 \\
\ldots
\end{array} \\
\llbracket N \to N \rrbracket &= q^0 \\
\begin{array}{c}
q_1 \\
0 \\
1 \\
\ldots
\end{array}
\end{align*}
\]

- Play = justified 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.

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

- Compositionality: \( \llbracket \text{succ} \ 7 \rrbracket = \llbracket \text{succ} \rrbracket; \llbracket 7 \rrbracket \)
Game-semantic Characterization of Safety

The variable binding restriction imposed by the safety constraint implies:

**Theorem**

- 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 $> \text{ord } m$.
- Conversely, 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.
The variable binding restriction imposed by the safety constraint implies:

**Theorem**
- 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 $\geq \text{ord } m$.
- Conversely, 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.
Safe PCF

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

Proposition

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

The first fully-abstract models of PCF were based on game semantics (Abramsky et al., Hyland and Ong, Nickau).

Question: Are P-i.j. strategies, suitably quotiented, fully abstract for Safe PCF?
Idealized Algol (IA) : Open problem

- **IA** = PCF + block-allocated variables + imperative features
- **IA**<sub>i</sub> + **Y**<sub>j</sub>: fragment of IA with finite base type, terms of order \( \leq i \), recursion limited to order \( j \)

Two IA terms are equivalent iff the two sets of complete plays of the game denotations are equal [Abramsky,McCusker].

- **IA**<sub>2</sub>: the set of complete plays is regular [Ghica&McCusker00].
- **IA**<sub>3</sub> + **Y**<sub>0</sub>: DPDA definable [Ong02].
- **IA**<sub>3</sub> + while: Visibly Pushdown Automaton definable [Murawski&Walukieicz05].

Hence observational equivalence is decidable for all these fragments. However at order 4, observational equivalence is undecidable [Mur05].

**Question**: Is observational equivalence decidable for the safe fragment of **IA**<sub>4</sub>?
**Conclusion and Future Works**

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

**Future work:**
- What is a (categorical) model of the safe lambda calculus?
- Can we obtain a fully abstract model of Safe PCF (with respect to safe contexts)?
- Complexity classes characterized 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).