The Safe λ-Calculus

William Blum and C.-H. Luke Ong

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

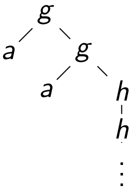
1. The safety restriction for higher-order grammars
2. The safe λ-calculus
3. Expressivity
4. Game-semantic characterization
5. Safe PCF, Safe IA

Higher-order grammars

Notation for types: $A_1 \to (A_2 \to (\ldots(A_n \to o))\ldots)$ 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 $\langle \Sigma, \mathcal{N}, \mathcal{R}, \mathcal{S} \rangle$ (terminals, non-terminals, rewritting rules, starting symbol)
- Example of a tree-generating order-2 grammar:

$$\begin{array}{rcl} S & \to & H\,a \\ H\,z^{o} & \to & F\,(g\,z) \\ F\,\phi^{(o,o)} & \to & \phi\,(\phi\,(F\,h)) \end{array}$$



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

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 $\mathrm{ord}\, A_1 \geq \mathrm{ord}\, A_2 \geq \cdots \geq \mathrm{ord}\, A_n$.

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 $h : o \to o$, $g : o \to o \to o$, $a : o$.
The following grammar is unsafe:

$$\begin{array}{rcl} S & \to & H\, a \\ H\, z^{o} & \to & F\,\underline{(g\, z)} \\ F\, \phi^{(o,o)} & \to & \phi\,(\phi\,(F\, h)) \end{array}$$

It is equivalent to the following safe grammar:

$$\begin{array}{rcl} S & \to & F(g\, a) \\ F\, \phi^{(o,o)} & \to & \phi\,(\phi\,(F\, h)) \end{array}$$

Safe grammars: examples

Take $h : o \to o$, $g : o \to o \to o$, $a : o$.
The following grammar is unsafe:

$$\begin{array}{rcl} S & \to & H\,a \\ H\,z^o & \to & F\,\underline{(g\,z)} \\ F\,\phi^{(o,o)} & \to & \phi\,(\phi\,(F\,h)) \end{array}$$

It is equivalent to the following safe grammar:

$$\begin{array}{rcl} S & \to & F(g\,a) \\ F\,\phi^{(o,o)} & \to & \phi\,(\phi\,(F\,h)) \end{array}$$

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 Caucal's result does not extend to unsafe grammars. However deciding μ-calculus theories is n-EXPTIME complete.

AdMO04 Proposed a notion of safety for the λ-calculus (unpublished).

Simply Typed λ-Calculus

- Simple types $A := o \mid A \to A$.
- The order of a type is given by $\mathrm{order}(o) = 0$, $\mathrm{order}(A \to B) = \max(\mathrm{order}(A) + 1, \mathrm{order}(B))$.
- Judgements of the form $\Gamma \vdash M : T$ where Γ is the context, M is the term and T is the type:

$$(var)\ \frac{}{x : A \vdash x : A} \qquad (wk)\ \frac{\Gamma \vdash M : A}{\Delta \vdash M : A}\ \Gamma \subset \Delta$$

$$(app)\ \frac{\Gamma \vdash M : A \to B \quad \Gamma \vdash N : A}{\Gamma \vdash MN : B} \qquad (abs)\ \frac{\Gamma, x : A \vdash M : B}{\Gamma \vdash \lambda x^A . M : A \to B}$$

- Example: $f : o \to o \to o, x : o \vdash (\lambda \varphi^{o \to o} x^o . \varphi\ x)(f\ x)$
- A single rule: β-reduction. e.g. $(\lambda x.M)N \to_\beta M[N/x]$

The Safe λ-Calculus

The formation rules

$$(var)\ \frac{}{x : A \vdash_s x : A} \qquad (wk)\ \frac{\Gamma \vdash_s M : A}{\Delta \vdash_s M : A}\ \Gamma \subset \Delta$$

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

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

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

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

Lemma

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

The Safe λ-Calculus : examples

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

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

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

- An example of safe term not in β-normal form:

$$(\lambda \varphi^{o\to o} x^o.\varphi\ x)(\lambda y^o.y)$$

The Safe λ-Calculus : examples

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

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

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

- An example of safe term not in β-normal form:

$$(\lambda \varphi^{o\to o} x^o.\varphi\ x)(\lambda y^o.y)$$

The Safe λ-Calculus : examples

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

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

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

- An example of safe term not in β-normal form:

$$(\lambda \varphi^{o\rightarrow o} x^o.\varphi\ x)(\lambda y^o.y)$$

The Safe λ-Calculus : examples

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

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

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

- An example of safe term not in β-normal form:

$$(\lambda \varphi^{o\rightarrow o} x^o.\varphi\ x)(\lambda y^o.y)$$

Variable Capture

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

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: $(\lambda x.(\lambda y.x))y \rightarrow_\beta (\lambda \underline{y}.x)[\underline{y}/x] \neq \lambda y.y$

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: $(\lambda x.(\lambda y.x))y \rightarrow_\beta (\lambda \underline{y}.x)[\underline{y}/x] \neq \lambda y.y$

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: $(\lambda x.(\lambda y.x))y \rightarrow_\beta (\lambda \underline{y}.x)[\underline{y}/x] \neq \lambda y.y$

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: $(\lambda x.(\lambda y.x))y \rightarrow_\beta (\lambda \underline{y}.x)[\underline{y}/x] \neq \lambda y.y$

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: $(\lambda x.(\lambda y.x))y \to_\beta (\lambda \underline{y}.x)[\underline{y}/x] \neq \lambda y.y$

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: examples

1. Contracting the β-redex in the following term

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

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, $\operatorname{ord} x = 0 \leq 1 = \operatorname{ord} f\ x$.

2. The term $(\lambda\varphi^{o\to o}x^{o}.\varphi\ x)(\lambda y^{o}.y)$ 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 β-contraction can be correctly implemented by capture permitting substitution, are safe.

Variable capture: examples

1. Contracting the β-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, $\mathrm{ord}\, x = 0 \leq 1 = \mathrm{ord}\, f\ x$.
2. The term $(\lambda \varphi^{o \to o} x^{o}.\varphi\ x)(\lambda y^{o}.y)$ 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 β-contraction can be correctly implemented by capture permitting substitution, are safe.

Variable capture: examples

1. Contracting the β-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, $\operatorname{ord} x = 0 \leq 1 = \operatorname{ord} f\ x$.

2. The term $(\lambda \varphi^{o \to o} x^o . \varphi\ x)(\lambda y^o . y)$ 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 β-contraction can be correctly implemented by capture permitting substitution, are safe.

Variable capture: examples

1. Contracting the β-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, $\mathrm{ord}\, x = 0 \leq 1 = \mathrm{ord}\, f \; x$.
2. The term $(\lambda \varphi^{o \to o} x^o . \varphi \; x)(\lambda y^o . y)$ 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 β-contraction can be correctly implemented by capture permitting substitution, are safe.

Transformations preserving safety

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

Transformations preserving safety

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

Transformations preserving safety

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

Transformations preserving safety

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

Transformations preserving safety

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

Transformations preserving safety

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

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:

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

Numerical functions (2)

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

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

In fact:

Theorem

Functions representable by safe λ-expressions of type $I \to \ldots \to I$ are exactly the multivariate polynomials.

Numerical functions (2)

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

- Natural number: $\overline{n} = \lambda sz.s^n z : (o \to o) \to o \to o$. Safe.
- Addition: $\overline{n+m} = \lambda \alpha^{(o,o)} x^o.(\overline{n}\,\alpha)(\overline{m}\,\alpha\,x)$. Safe.
- Multiplication: $\overline{n.m} = \lambda \alpha^{(o,o)}.\overline{n}\,(\overline{m}\,\alpha)$. 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 λ-expressions of type $I \to \ldots \to I$ are exactly the multivariate polynomials.

Numerical functions (2)

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

- Natural number: $\overline{n} = \lambda sz.s^n z : (o \to o) \to o \to o$. Safe.
- Addition: $\overline{n+m} = \lambda \alpha^{(o,o)} x^o.(\overline{n}\,\alpha)(\overline{m}\,\alpha\,x)$. Safe.
- Multiplication: $\overline{n.m} = \lambda \alpha^{(o,o)}.\overline{n}\,(\overline{m}\,\alpha)$. 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 λ-expressions of type $I \to \ldots \to I$ are exactly the multivariate polynomials.

Numerical functions (2)

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

- Natural number: $\overline{n} = \lambda sz.s^n z : (o \to o) \to o \to o$. Safe.
- Addition: $\overline{n+m} = \lambda \alpha^{(o,o)} x^o.(\overline{n}\,\alpha)(\overline{m}\,\alpha\,x)$. Safe.
- Multiplication: $\overline{n.m} = \lambda \alpha^{(o,o)}.\overline{n}\,(\overline{m}\,\alpha)$. 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 λ-expressions of type $I \to \ldots \to I$ are exactly the multivariate polynomials.

Numerical functions (2)

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

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

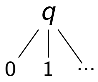
In fact:

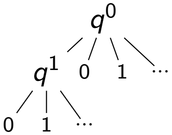
Theorem

Functions representable by safe λ-expressions of type $I \to \ldots \to I$ 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
  $[\![\mathbb{N}]\!] =$ 

  $[\![\mathbb{N} \to \mathbb{N}]\!] =$ 

- 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 $[\![M]\!]$, is a strategy for P.
- $[\![7 : \mathbb{N}]\!] = \{\epsilon, q, q\ 7\}$
  $[\![\mathrm{succ} : \mathbb{N} \to \mathbb{N}]\!] = \mathit{Pref}(\{q^0 q^1 n(n+1) \mid n \in \mathbb{N}\})$
- Compositionality: $[\![\mathrm{succ}\ 7]\!] = [\![\mathrm{succ}]\!] ; [\![7]\!]$

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 $> \operatorname{ord} m$.
- Conversely, if a *closed* term is denoted by a P-incrementally justified strategy then its η-long β-normal form is safe.

Corollary

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

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 $> \mathrm{ord}\, m$.
- Conversely, if a *closed* term is denoted by a P-incrementally justified strategy then its η-long β-normal form is safe.

Corollary

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

Safe PCF

- 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
- Introduced by John Reynolds, 1997.
- $IA_i + Y_j$: 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_2: the set of complete plays is regular [Ghica&McCusker00].
- $IA_3 + Y_0$: DPDA definable [Ong02].
- $IA_3 + \mathit{while}$: Visibly Pushdown Automaton definable [Murawski&Walukievicz05].

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_4?

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 λ-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).