The Safe λ-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 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

Part I The safety restriction

1. Safety for higher-order grammars
2. The safe λ-calculus
3. Expressivity

Part II Game-semantic

1. The Correspondence Theorem
2. Game-semantic characterisation
3. Compositionality

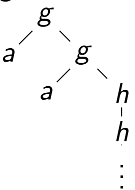
Part I : The Safety Restriction

Higher-order grammars

Notation for types: $A_1 \to (A_2 \to (\ldots (A_n \to o)) \ldots)$ 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 $\langle \Sigma, \mathcal{N}, \mathcal{R}, \mathcal{S} \rangle$ (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, ..., 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.

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:

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

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

- 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 $\forall y \in \Gamma : \mathrm{ord}\, y \geq \mathrm{ord}\, B$

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

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

Lemma

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

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

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

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.

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. 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, β-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 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. 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, β-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 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. 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, β-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 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. 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, β-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 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. 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, β-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 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. 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, β-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.

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 $\not\vdash_s \lambda x^o.(\lambda y^o z^o.y)x : (o, o, o)$.
- 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 $\not\vdash_s \lambda x^o.(\lambda y^o z^o.y) x : (o, o, o)$.
- 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 $\nvdash_s \lambda x^o.(\lambda y^o z^o.y)x : (o, o, o)$.
- 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 $\not\vdash_s \lambda x^o.(\lambda y^o z^o.y)x : (o, o, o)$.
- 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 $\not\vdash_s \lambda x^o.(\lambda y^o z^o.y)x : (o, o, o)$.
- 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 $\nvdash_s \lambda x^o.(\lambda y^o z^o.y)x : (o, o, o)$.
- η-long normal expansion preserves safety.

Expressivity

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

- de Miranda 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:

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

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

So *cond* is not representable in the Safe λ-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:

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

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

Theorem

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

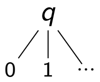
So *cond* is not representable in the Safe λ-calculus and C is unsafe.

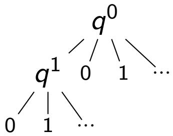
Part II : Game semantics

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

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

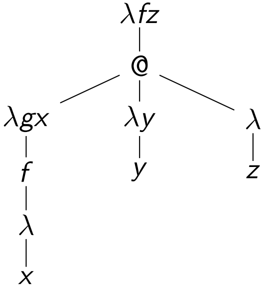
  $[\![\mathbb{N} \to \mathbb{N}]\!] =$ 

- 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 $[\![M]\!]$, 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: $[\![\mathrm{succ}\ 7]\!] = [\![\mathrm{succ}]\!] ; [\![7]\!]$

Computation trees and traversals

Computation tree: AST of the η-long normal form of a term.

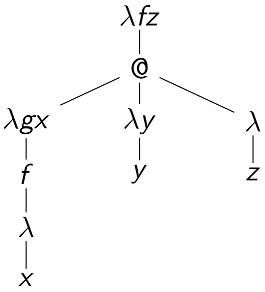
Example: $M \equiv \lambda fz.(\lambda gx.fx)(\lambda y.y)z$ of type $(o \to o) \to o \to o$.



Computation trees and traversals

Computation tree: AST of the η-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.

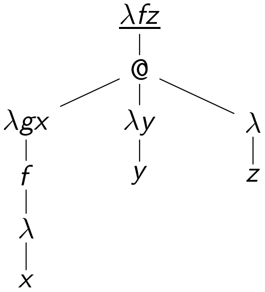


Computation trees and traversals

Computation tree: AST of the η-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.

$t = \lambda fz$

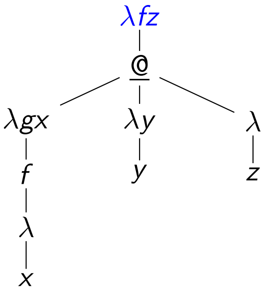


Computation trees and traversals

Computation tree: AST of the η-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.

$$t = \lambda fz \cdot @$$

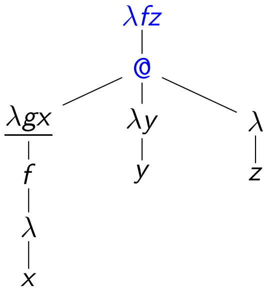


Computation trees and traversals

Computation tree: AST of the η-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.

$$t = \lambda fz \cdot @ \cdot \lambda gx$$

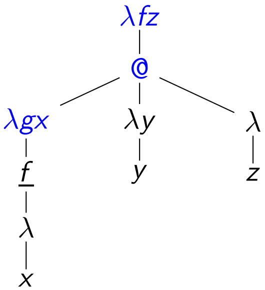


Computation trees and traversals

Computation tree: AST of the η-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.

$$t = \lambda fz \cdot @ \cdot \lambda gx \cdot f$$

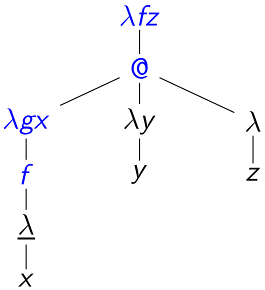


Computation trees and traversals

Computation tree: AST of the η-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.

$$t = \lambda fz \cdot @ \cdot \lambda gx \cdot f \cdot \lambda$$

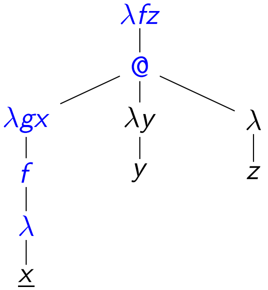


Computation trees and traversals

Computation tree: AST of the η-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.

$$t = \lambda fz \cdot @ \cdot \lambda gx \cdot f \cdot \lambda \cdot x$$

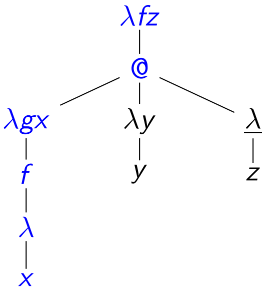


Computation trees and traversals

Computation tree: AST of the η-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.

$$t = \lambda fz \cdot @ \cdot \lambda gx \cdot f \cdot \lambda \cdot x \cdot \lambda$$

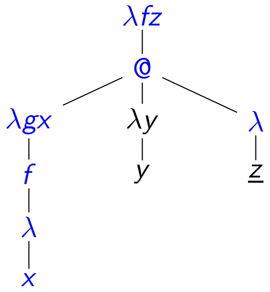


Computation trees and traversals

Computation tree: AST of the η-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.

$$t = \lambda fz \cdot @ \cdot \lambda gx \cdot f \cdot \lambda \cdot x \cdot \lambda \cdot z$$



Computation trees and traversals

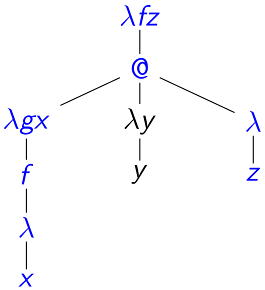
Computation tree: AST of the η-long normal form of a term.
Example: $M \equiv \lambda fz.(\lambda gx.fx)(\lambda y.y)z$ of type $(o \to o) \to o \to o$.

Traversal: justified sequence of nodes representing the computation.

$$t = \lambda fz \cdot @ \cdot \lambda gx \cdot f \cdot \lambda \cdot x \cdot \lambda \cdot z$$

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

$$t \upharpoonright r = \lambda fz \cdot f \cdot \lambda \cdot z$$



Computation trees and traversals

Computation tree: AST of the η-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.

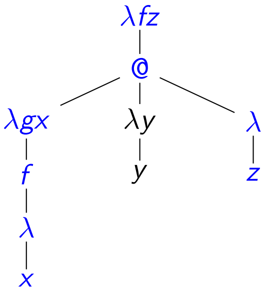
$$t = \lambda fz \cdot @ \cdot \lambda gx \cdot f \cdot \lambda \cdot x \cdot \lambda \cdot z$$

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

$$t \upharpoonright r = \lambda fz \cdot f \cdot \lambda \cdot z$$

@-nodes removal:

$$t - @ = \lambda fz \cdot \lambda gx \cdot f \cdot \lambda \cdot x \cdot \lambda \cdot z$$



The Correspondence Theorem

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

$$\varphi : \mathcal{T}rav(M)^{-@} \xrightarrow{\cong} \langle\!\langle M \rangle\!\rangle$$

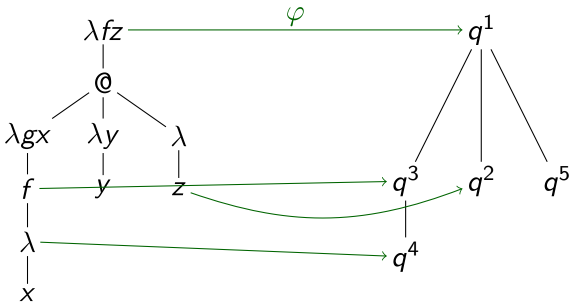
$$\varphi : \mathcal{T}rav(M)^{\upharpoonright r} \xrightarrow{\cong} [\![M]\!] \ .$$

where

- $\mathcal{T}rav(M)$ = set of traversals of the computation tree of M
- $\mathcal{T}rav(M)^{\upharpoonright r} = \{t \upharpoonright r \mid t \in \mathcal{T}rav(M)\}$
- $\mathcal{T}rav(M)^{-@} = \{t - @ \mid t \in \mathcal{T}rav(M)\}$
- $[\![M]\!]$ = game-semantic denotation of M
- $\langle\!\langle M \rangle\!\rangle$ = revealed denotion (*i.e.* internal moves are uncovered.)

The Correspondence Theorem (example)

Left: computation tree. Right: arena.



Take the traversal $t = \lambda fz \cdot @ \cdot \lambda gx \cdot f \cdot \lambda \cdot x \cdot \lambda \cdot z$. We have: $\varphi(t \upharpoonright r) = \varphi(\lambda fz \cdot f \cdot \lambda \cdot z) = q^1\ q^3\ q^4\ q^2 \in [\![M]\!]$.

The Correspondence Theorem (2)

Computation tree notions	Game-semantic equivalents
computation tree	arena(s)
traversal	uncovered play
reduced traversal	play
path in the computation tree	P-view of an uncovered play

Game-semantic Characterisation of Safety

- The computation tree of a safe term is incrementally-bound : each variable x is bound by the first λ-node occurring in the path to the root with order $> \operatorname{ord} x$.
- 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 $> \operatorname{ord} m$.
- Reciprocally, 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 Characterisation of Safety

- The computation tree of a safe term is incrementally-bound : each variable x is bound by the first λ-node occurring in the path to the root with order $> \operatorname{ord} x$.
- 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 $> \operatorname{ord} m$.
- Reciprocally, 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 Characterisation of Safety

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

Proposition

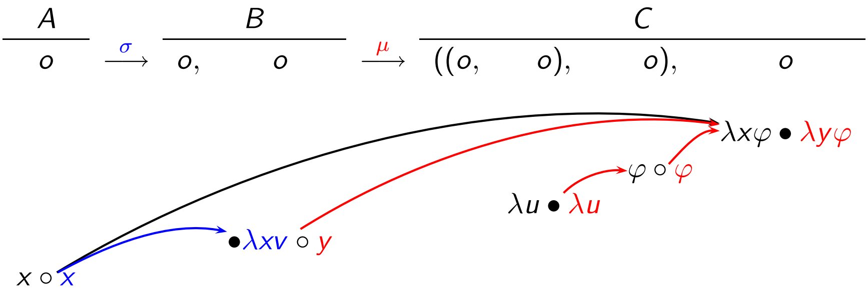
- Safe terms are denoted by P-incrementally justified strategies: each P-move m points to the last O-move in the P-view with order $> \operatorname{ord} m$.
- Reciprocally, if a *closed* term is denoted by a P-incrementally justified strategy then its η-long β-normal form is safe.

Corollary

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

Compositionality

Question Do P-incrementally-justified strategies compose?
No. Take $\sigma = [\![\vdash_s \lambda x^o v^o.x : o \to (o,o)]\!]$ and $\mu = [\![\vdash_s \lambda y^{(o,o)} \varphi^{((o,o),o)}.\varphi(\lambda u^o.ya) : (o,o) \to (((o,o),o),o)]\!]$ for some constant $a : o$. We have $\sigma \fatsemi \mu = [\![\lambda x\varphi.\varphi(\underline{\lambda u.x})]\!]$ which is not P-i.j. by the previous proposition.



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 σ we have $\mathrm{ord}_A\, m \geq \mathrm{ord}\, B$.

- 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 λ-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 σ we have $\mathrm{ord}_A\, m \geq \mathrm{ord}\, B$.

- 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 λ-calculus.

Safe PCF

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

Proposition

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

Definability

Let σ be a well-bracketed innocent P-i.j. strategy with finite view function defined on a PCF arena $A_1 \times \ldots \times A_i \to B$. σ 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 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).