The Safe λ-Calculus

William Blum

Oxford University Computing Laboratory

PRG Student Conference

13 October 2006

Overview

- Safety is a restriction for higher-order grammars.
- It can be transposed to the λ-calculus, giving rise to the Safe λ-calculus.
- Safety has nice algorithmic properties, automata-theoretic and game-semantic characterizations.

What is 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.

Theorem (Knapik, Niwiński and Urzyczyn (2001,2002))

1. *The Monadic Second Order (MSO) model checking problem for trees generated by safe higher-order grammars of any order is decidable.*
2. *Automata-theoretic characterization: Safe grammars of order n are as expressive as pushdown automata of order n.*

- Aehlig, de Miranda, Ong (2004) introduced the Safe λ-calculus.

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. $(\lambda x.M)N \to_\beta M[N/x]$

Simply Typed λ-Calculus

- Simple types $A := o \mid A \to A$.
- The order of a type is given by $\text{order}(o) = 0$, $\text{order}(A \to B) = \max(\text{order}(A) + 1, \text{order}(B))$.
- Judgements of the form $\Gamma \vdash M : T$ where 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. $(\lambda x.M)N \to_\beta M[N/x]$

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. $(\lambda x.M)N \to_\beta M[N/x]$

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. $(\lambda x.M)N \to_\beta M[N/x]$

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. $(\lambda x.M)N \to_\beta M[N/x]$

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: switch to 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.

Safety avoids the need for variable renaming!

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: switch to 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.

Safety avoids the need for variable renaming!

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: switch to 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.

Safety avoids the need for variable renaming!

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: switch to 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.

Safety avoids the need for variable renaming!

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: switch to 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.

Safety avoids the need for variable renaming!

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: switch to 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.

Safety avoids the need for variable renaming!

The Safe λ-Calculus

The formation rules

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

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

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 $\forall y \in \Gamma : \mathrm{ord}(y) \geq \mathrm{ord}(A_1 \to \ldots \to A_n \to B)$

Property

In the Safe λ-calculus there is no need to rename variables when performing β-reduction.

The Safe λ-Calculus

The formation rules

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

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

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 $\forall y \in \Gamma : \mathrm{ord}(y) \geq \mathrm{ord}(A_1 \to \ldots \to A_n \to B)$

Property

In the Safe λ-calculus there is no need to rename variables when performing β-reduction.

Example

- 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, $\mathrm{ord}(x) = 0 \leq 1 = \mathrm{ord}(f \; x)$.

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

Example

- 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, $\mathrm{ord}(x) = 0 \leq 1 = \mathrm{ord}(f\ x)$.

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

Example

- 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, $\mathrm{ord}(x) = 0 \leq 1 = \mathrm{ord}(f\ x)$.
- The term $(\lambda \varphi^{o \to o} x^{o}.\varphi\ x)(\lambda y^{o}.y)$ is safe.

The Correspondence Theorem

Let $M : T$ be a pure simply typed term.

- Game-semantics provides a model of λ-calculus. M is denoted by a strategy $[\![M]\!]$ on a game induced by T.
- A strategy is represented by a set of sequences of moves together with links (each move points to a preceding move).
- Computation tree = canonical tree representation of a term.
- Traversals $\mathcal{T}rav(M)$ = sequences of nodes with links respecting some formation rules.

The game semantics of a term can be represented on the computation tree:

$$\mathcal{T}rav(M) \cong \langle\!\langle M \rangle\!\rangle$$

$$Reduction(\mathcal{T}rav(M)) \cong [\![M]\!]$$

where $\langle\!\langle M \rangle\!\rangle$ is the revealed game-semantic denotion (i.e. internal moves are uncovered).

The Correspondence Theorem

Let $M : T$ be a pure simply typed term.

- Game-semantics provides a model of λ-calculus. M is denoted by a strategy $[\![M]\!]$ on a game induced by T.
- A strategy is represented by a set of sequences of moves together with links (each move points to a preceding move).
- Computation tree = canonical tree representation of a term.
- Traversals $\mathcal{T}rav(M)$ = sequences of nodes with links respecting some formation rules.

The game semantics of a term can be represented on the computation tree:

$$\mathcal{T}rav(M) \cong \langle\!\langle M \rangle\!\rangle$$

$$Reduction(\mathcal{T}rav(M)) \cong [\![M]\!]$$

where $\langle\!\langle M \rangle\!\rangle$ is the revealed game-semantic denotion (i.e. internal moves are uncovered).

The Correspondence Theorem

Let $M : T$ be a pure simply typed term.

- Game-semantics provides a model of λ-calculus. M is denoted by a strategy $[\![M]\!]$ on a game induced by T.
- A strategy is represented by a set of sequences of moves together with links (each move points to a preceding move).
- Computation tree = canonical tree representation of a term.
- Traversals $\mathcal{T}rav(M)$ = sequences of nodes with links respecting some formation rules.

The game semantics of a term can be represented on the computation tree:

$$\mathcal{T}rav(M) \cong \langle\!\langle M \rangle\!\rangle$$

$$Reduction(\mathcal{T}rav(M)) \cong [\![M]\!]$$

where $\langle\!\langle M \rangle\!\rangle$ is the revealed game-semantic denotion (i.e. internal moves are uncovered).

Game-semantic Characterisation of Safety

- Computation tree of safe terms are incrementally-bound : each variable x is bound by the first λ node occurring in *the path to the root* with order $>$ ord(x).
- By the Correspondence theorem, this implies that safe terms are denoted by incrementally-justified strategies: each move m points to the last other player's move with order $>$ ord(m).

Corollary

Justification pointers are redundant in the game-semantics of safe terms. Hence the game semantics of a safe term has a succinct representation.

Game-semantic Characterisation of Safety

- Computation tree of safe terms are incrementally-bound : each variable x is bound by the first λ node occurring in *the path to the root* with order $> \mathrm{ord}(x)$.
- By the Correspondence theorem, this implies that safe terms are denoted by incrementally-justified strategies: each move m points to the last other player's move with order $> \mathrm{ord}(m)$.

Corollary

Justification pointers are redundant in the game-semantics of safe terms. Hence the game semantics of a safe term has a succinct representation.

Game-semantic Characterisation of Safety

- Computation tree of safe terms are incrementally-bound : each variable x is bound by the first λ node occurring in *the path to the root* with order $> \mathrm{ord}(x)$.
- By the Correspondence theorem, this implies that safe terms are denoted by incrementally-justified strategies: each move m points to the last other player's move with order $> \mathrm{ord}(m)$.

Corollary

Justification pointers are redundant in the game-semantics of safe terms. Hence the game semantics of a safe term has a succinct representation.

Conclusion and Further Work

Conclusion:

Safety is a syntactic constraint with nice algorithmic and game-semantic properties.

Related works:

- Forthcoming thesis of Jolie G. de Miranda about unsafety.
- Ong introduced computation trees in LICS2006 to prove decidability of MSO theory on infinite trees generated by higher-order grammars (whether safe or not).
- Stirling recently proved decidability of higher-order pattern matching with a game-semantic approach relying on equivalent notions of computation tree and traversal.

Open questions:

- Complexity classes characterised with the Safe λ-calculus?
- Does the pointer economy extend to Safe Idealized Algol? Decidability of contextual equivalence?

Bibliography

Samson Abramsky and Guy McCusker.
Game semantics, Lecture notes.
In *Proceedings of the 1997 Marktoberdorf Summer School.*
Springer-Verlag, 1998.

Klaus Aehlig, Jolie G. de Miranda, and C.-H. Luke Ong.
Safety is not a restriction at level 2 for string languages.
Technical report. University of Oxford, 2004.

C.-H. Luke Ong.
On model-checking trees generating by higher-order recursion schemes.
In *Proceedings of LICS.* Computer Society Press, 2006.

Colin Stirling
A Game-Theoretic Approach to Deciding Higher-Order Matching.
In *Proceedings of ICALP.* Springer, 2006.