Verifying Finitely-Presentable Infinite Structures:

A Game-Semantic Approach

(Lecture 1)

Luke Ong

University of Oxford

29 May - 2 June 2006

Motivation

Verification and (Game) Semantics

What is the Model Checking approach to Verification?

Given a system (e.g. lift controller, operating system) and a desired property (e.g. deadlock freedom, liveness) of the system:

1. Construct an abstract model M of the system.
2. Describe the property as a formula φ in some logic $\mathcal{L}$.
3. Exhaustively check the model M for violation of φ.

Extremely successful in verifying relatively “flat, unstructured” finite-state processes (e.g. protocols, circuits); less effective when applied to software.

Key (interdependent) semantic and algorithmic questions:

- Does M model the system accurately?
- Is the problem “Does M satisfy φ?” decidable?
- Is the violation check efficient (or better, optimal)?

Our approach is to analyse basic problems in Verification using (game-)semantic methods.

Overview 1: Trees generated by recursion schemes

A Basic Problem in Model Checking: Find classes of finitely-presentable infinite structures with decidable monadic second-order (MSO) theories.

We study the infinite hierarchy of (possibly infinite) term-trees generated by higher-order recursion schemes (= simply-typed lambda calculus + uninterpreted 1st-order function symbols + fixpoints).

Why?

- Natural case study of the game-semantic approach.
- Rich and unifying tree hierarchy - subsumes major classes.
- Robust framework - admits several different characterizations.

Theorem. For each $n \geq 0$, the modal mu-calculus model checking-problem for ***RecSchTree***$_n$ (i.e. trees generated by order-n recursion schemes) is n-EXPTIME complete. Thus these trees have decidable MSO theories.

Overview 2: From Trees to Graphs

Characterising expressiveness of higher-order recursion schemes:
Order-n Collapsible Pushdown Automata (CPDA)

- Each stack symbol in n-stack "remembers" the stack content at the point it was first created (i.e. pushed).
- collapse (= panic) collapses the n-stack up to the point as remembered by the top element of the stack.

Theorem. As tree-generating (resp. graph-generating) devices, order-n recursion schemes = order-n collapsible pushdown automata, for each $n \geq 0$.

The same game-semantic approach is just as effective a basis for **model-checking (new?) hierarchies of graphs**.
E.g. Solving parity games over order-n (collapsible) pushdown graphs.

Many further directions, and open problems.

Outline

Higher-order Recursion Schemes

Order of a Type

Types are ranged over by $A, B, \cdots$.

$$A \quad ::= \quad o \quad | \quad (A \to B)$$

Every type can be written uniquely as

$$A_1 \to (A_2 \cdots \to (A_n \to o) \cdots), \quad n \geq 0$$

which is abbreviated to $A_1 \to A_2 \cdots \to A_n \to o$. (Convention: arrows associate to the right.)

The **order** of a type measures how nested it is on the LHS of the arrow.

$$\begin{aligned} order(o) &= 0 \\ order(A \to B) &= \max(order(A) + 1, order(B)) \end{aligned}$$

Notation. $\quad e : A \quad$ means "expression e has type A".

Example of an order-1 recursion scheme

Everything is typed!

Ranked alphabet of **terminals**: $\Sigma = \{ \underline{f}, \underline{g}, \underline{a} \}$ with

$$\underline{f} : o \to o, \quad \underline{g} : o \to o, \quad \underline{a} : o$$

A finite system of well-typed **rewrite rules**:

$$G_1 \; : \; \begin{cases} S & = & F\,\underline{a} \\ F\,x & = & \underline{f}\,x\,(F\,(\underline{g}\,x)) \end{cases}$$

Each rule is a recursive definition of a **non-terminal** (upper letters S and F).

Order-$(n+1)$ non-terminals are defined with the help of **variables** of order up to order n.

Order-n (deterministic) recursion scheme $G = (\mathcal{N}, \Sigma, \mathcal{R}, S)$

Fix a set of typed variables (written as φ, x, y etc).

- $\mathcal{N}$: Typed non-terminals of order at most n (written as upper-case letters), including a distinguished start symbol $S : o$.
- Σ: **Ranked** alphabet of terminals: $\underline{f} \in \Sigma$ has arity $ar(\underline{f}) \geq 0$ which determines a first-order type $\underline{f} : \underbrace{o \to \cdots \to o}_{ar(\underline{f})} \to o$
- $\mathcal{R}$: An equation for each non-terminal $D : A_1 \to \cdots \to A_m \to o$ of shape

$$D\,\varphi_1 \cdots \varphi_m \quad = \quad e$$

  where the term $e : o$ is constructed from
  - terminals $\underline{f}, \underline{g}, \underline{a}$, etc. from Σ
  - variables $\varphi_1 : A_1, \cdots, \varphi_m : A_m$ from Var,
  - non-terminals D, F, G, etc. from $\mathcal{N}$.

  using the application rule: If $s : A \to B$ and $t : A$ then $(st) : B$.

Examples

Set $\Sigma = \{ \underline{f}, \underline{f'} : o^2 \to o,\ \underline{g} : o \to o,\ \underline{a} : o \}$.

1. An order-0 example: No variables!

$$G_1 \ : \ \left\{ \begin{array}{rcl} S & = & \underline{f}\,T\,T \\ T & = & \underline{f'}\,U\,U \\ U & = & \underline{f}\,T\,T \end{array} \right.$$

2. An order-2 example.

$B : (o \to o) \to (o \to o) \to o \to o, \quad F : (o \to o) \to o$

$$G_2 \ : \ \left\{ \begin{array}{rcl} S & = & F\,\underline{g} \\ B\,\varphi\,\psi\,x & = & \varphi\,(\psi\,x) \\ F\,\varphi & = & \underline{f}\,(\varphi\,\underline{a})\,(F\,(B\,\varphi\,\varphi)) \end{array} \right.$$

Tree Generated by a Recursion Scheme G

The ***value tree*** $[\![\, G \,]\!]$ of a recursion scheme G is a possibly infinite applicative term *constructed from the terminals*, which is obtained by unfolding the equations *ad infinitum*, replacing formal by actual parameters each time, starting from S.

Example. $\Sigma = \{\, \underline{f}, \underline{g}, \underline{a} \,\}$. Take

$$G_1 \;:\; \begin{cases} \quad S & = \quad F\, \underline{a} \\ F\, x & = \quad \underline{f}\, x\, (F\, (\underline{g}\, x)) \end{cases}$$

Thus

$$\begin{array}{rcl} S & \rightarrow & F\, \underline{a} \\ & \rightarrow & \underline{f}\, \underline{a}\, (F\, (\underline{g}\, \underline{a})) \\ & \rightarrow & \underline{f}\, \underline{a}\, (\underline{f}\, (\underline{g}\, \underline{a})\, (F\, (\underline{g}\, (\underline{g}\, \underline{a})))) \\ & \rightarrow & \cdots \end{array}$$

We have $[\![\, G_1 \,]\!] = \underline{f}\, \underline{a}\, (\underline{f}\, (\underline{g}\, \underline{a})\, (\underline{f}\, (\underline{g}\, (\underline{g}\, \underline{a}))(\cdots)))$.

Tree Generated by a Recursion Scheme G

The ***value tree*** $[\![G]\!]$ of a recursion scheme G is a possibly infinite applicative term *constructed from the terminals*, which is obtained by unfolding the equations *ad infinitum*, replacing formal by actual parameters each time, starting from S.

Example. $\Sigma = \{ \underline{f}, \underline{g}, \underline{a} \}$. Take

$$G_1 \; : \; \begin{cases} S & = & F \, \underline{a} \\ F \, x & = & \underline{f} \, x \, (F \, (\underline{g} \, x)) \end{cases}$$

We have $[\![G_1]\!] = \underline{f} \, \underline{a} \, (\underline{f} \, (\underline{g} \, \underline{a}) \, (\underline{f} \, (\underline{g} \, (\underline{g} \, \underline{a}))(\cdots)))$.

We view the infinite term $[\![G]\!]$ as a **Σ-labelled (ranked and ordered) tree** (generated by G).

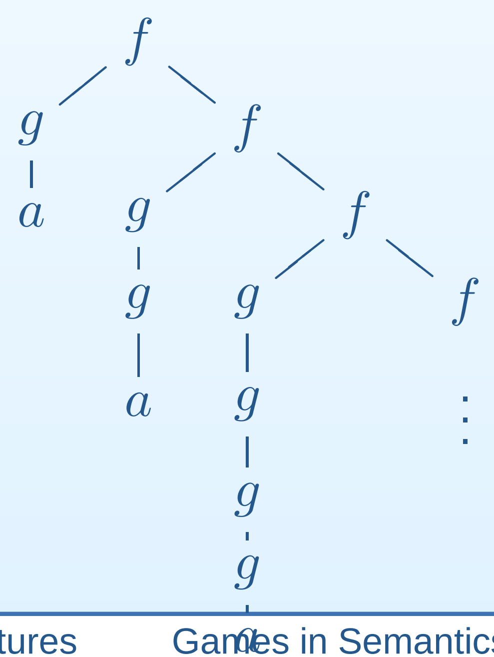
Formally a ***Σ-labelled tree*** is a function $t : \mathrm{dom}(t) \longrightarrow \Sigma$ such that $\mathrm{dom}(t) \subseteq \{ 1, \cdots, m \}^*$ is prefix-closed, and for all nodes $\alpha \in T$, the Σ-symbol $t(\alpha) \in \Sigma$ has arity k iff α has k children, namely $\alpha \, 1, \cdots, \alpha \, k \in T$.

An order-2 example

$\Sigma = \{\, \underline{f}, \underline{g}, \underline{a} \,\}$. $B : (o \to o) \to (o \to o) \to o \to o, \quad F : (o \to o) \to o$

$$G_2 \;:\; \left\{ \begin{array}{rcl} S & = & F\,\underline{g} \\ B\,\varphi\,\psi\,x & = & \varphi\,(\psi\,x) \\ F\,\varphi & = & \underline{f}\,(\varphi\,\underline{a})\,(F\,(B\,\varphi\,\varphi)) \end{array} \right.$$

The ***value tree***, $[\![\, G_2 \,]\!] : \{\, 1, 2 \,\}^* \longrightarrow \Sigma$, is: $\left\{ \begin{array}{ccccccc} \epsilon & \mapsto & f & \quad & 11 & \mapsto & a \\ 1 & \mapsto & g & & 21 & \mapsto & g \\ 2 & \mapsto & f & & 22 & \mapsto & f \\ & \cdots & & & \cdots & & \end{array} \right.$



A Model-Checking Problem

MSO model-checking problem for trees

For $n \geq 0$ write ***RecSchTree***$_n$ for the class of Σ-labelled trees generated by order-n recursion schemes.

Fact:

$$\begin{aligned} \textbf{\textit{RecSchTree}}_0 &= \{ \text{regular trees (i.e. generated by finite automata)} \} \\ \textbf{\textit{RecSchTree}}_1 &= \{ \text{algebraic trees (i.e. generated by DPDA)} \} \end{aligned}$$

MSO MODEL-CHECKING PROBLEM FOR ***RecSchTree***$_n$

- INSTANCE: An order-n recursion scheme G, and an MSO formula φ
- QUESTION: Does the Σ-labelled tree $[\![G]\!]$ satisfy φ?

Two problems about the tree hierarchy $\langle$ *RecSchTree*$_n$ $\rangle_{n \in \omega}$

1. **Decidability**. For which $n \geq 2$ is the problem decidable?
2. Find **automata-theoretic characterization** of $\langle$ ***RecSchTree***$_n$ $\rangle_{n \in \omega}$.

We use game semantics to solve the problems.

Representing Σ-labelled trees as logical structures

Take a Σ-labelled tree $t : \mathrm{dom}(t) \longrightarrow \Sigma$.
Represent t by the tuple

$$\langle\, \mathrm{dom}(t), \quad \langle\, \mathbf{d}_i : 1 \leq i \leq m \,\rangle, \quad \langle\, \mathbf{p}_f : f \in \Sigma \,\rangle\,\rangle$$

where

- $\mathrm{dom}(t) \subseteq \{\, 1, \cdots, m \,\}^*$ with $m = \max\{\, ar(f) : f \in \Sigma \,\}$
- Parent-child relationship: $\mathbf{d}_i = \{\, (\alpha, \alpha i) : \alpha \in \mathrm{dom}(t) \ \wedge \ \alpha i \in \mathrm{dom}(t) \,\}$
- Node labelling: $\mathbf{p}_f = \{\, \alpha \in \mathrm{dom}(t) : t(\alpha) = f \,\}$.

Hence given a ranked alphabet Σ, fix a vocabulary with binary predicate symbols $\mathbf{d}_i$ where $1 \leq i \leq$ maximum arity of Σ-symbols, and unary predicate symbols $\mathbf{p}_f$, one for each $f \in \Sigma$.

Monadic Second-Order Logic (for Σ-labelled trees)

First-order variables: x, y, z, etc. (ranging over *nodes*, which are finite words over $\{1, \cdots, m\}$, for a fixed m)

Second-order variables: X, Y, Z, etc. (ranging over *sets* of nodes i.e. *monadic* relations)

MSO formulas are built up from **atomic formulas**:

1. Parent-child relationship between nodes: $\mathbf{d}_i(x, y) \equiv$ "y is i-child of x"
2. Node labelling: $\mathbf{p}_f(x) \equiv$ "x has label f" where f is a Σ-symbol
3. Set-membership: $x \in X$

and closed under

- boolean connectives: $\neg$, $\vee$ etc.
- first-order quantifications: $\forall x.-, \exists x.-$
- second-order quantifications: $\forall X.-, \exists X.-$.

Why MSO Logic?

It is a kind of **gold standard**!

- MSO is *very* expressive. Over graphs, MSO is strictly more expressive than the modal mu-calculus, into which all standard temporal logics (e.g. LTL, CTL, CTL$*$, etc.) can embed.
  Over trees, modal mu-calculus is as expressive as (but algorithmically more tractable than) MSO: For every MSO φ, there is a modal mu-calculus formula p_φ s.t. for every Σ-labelled tree t, we have $t \models \varphi \iff t, \epsilon \models p_\varphi$.
- Any obvious extension of MSO would break decidability. Either of the following would permit an encoding of a Turing machine:
  - Second-order quantification over binary relations.
  - Freely interpretable binary relations in the vocabulary.

  E.g. $T_a(i, t)$ = "i-th cell of the semi-infinite tape contains $a \in \Sigma$ at time t".

Examples of MSO-definable properties

Several useful relations are definable:

1. Set inclusion (and hence equality): $X \subseteq Y \;\equiv\; \forall x \,.\, x \in X \rightarrow x \in Y$.
2. "Is-an-ancestor-of" or prefix ordering $x \leq y$ (and hence $x = y$):

$$\begin{aligned} \mathsf{PrefCl}(X) \;&\equiv\; \forall xy \,.\, y \in X \wedge \bigvee_{i=1}^{m} \mathbf{d}_i(x, y) \;\rightarrow\; x \in X \\ x \leq y \;&\equiv\; \forall X \,.\, \mathsf{PrefCl}(X) \wedge y \in X \;\rightarrow\; x \in X \end{aligned}$$

Reachability property: "X is a path"

$$\begin{aligned} \mathsf{Path}(X) \;&\equiv\; \forall xy \in X \,.\, x \leq y \vee y \leq x \\ &\wedge\; \forall xyz \,.\, x \in X \wedge z \in X \wedge x \leq y \leq z \;\rightarrow\; y \in X \end{aligned}$$

$\mathsf{MaxPath}(X) \equiv \mathsf{Path}(X) \wedge \forall Y \,.\, \mathsf{Path}(Y) \wedge X \subseteq Y \;\rightarrow\; Y \subseteq X$.

MSO is expressive: more examples

Recurrence Property

A set of nodes is a **cut** if no two nodes in it are $\leq$-compatible, and it has a non-empty intersection with every maximal path.

$$\begin{aligned} \mathsf{Cut}(X) \;&\equiv\; \forall xy \in X\,.\,\neg(x \leq y \,\vee\, y \leq x) \\ &\wedge\; \forall Z\,.\,\mathsf{MaxPath}(Z) \;\rightarrow\; \exists z \in Z\,.\,z \in X \end{aligned}$$

Fact. A set X of nodes in a finitely-branching tree is finite iff there is a cut C such that every X-node is a prefix of some C-node.

$$\mathsf{Finite}(X) \;\equiv\; \exists Y\,.\,\mathsf{Cut}(Y) \;\wedge\; \forall x \in X\,.\,\exists y \in Y\,.\,x \leq y$$

Hence “there are finitely many nodes labelled by f” is expressible in MSO by

$$\exists X\,.\,\mathsf{Finite}(X) \;\wedge\; \forall x\,.\,\mathbf{p}_f(x) \rightarrow x \in X$$

But “MSO cannot count”: E.g. “X has twice as many elements as Y”.

Infinite Structures with Decidable MSO Theories

Structures with decidable MSO theories: some milestones

[In timeline below, each item subsumes developments in preceding items.]

1. Rabin 1969: Regular trees. “Mother of all decidability results”

Structures with decidable MSO theories: some milestones

[In timeline below, each item subsumes developments in preceding items.]

1. Rabin 1969: Regular trees. “Mother of all decidability results”
2. Muller and Schupp 1985: Configuration graphs of pushdown automata.

Structures with decidable MSO theories: some milestones

[In timeline below, each item subsumes developments in preceding items.]

1. Rabin 1969: Regular trees. “Mother of all decidability results”
2. Muller and Schupp 1985: Configuration graphs of pushdown automata.
3. Caucal (ICALP 1996): Prefix-recognizable graphs (= ϵ-closures of configuration graphs of pushdown automata, Stirling 2000).

Structures with decidable MSO theories: some milestones

[In timeline below, each item subsumes developments in preceding items.]

1. Rabin 1969: Regular trees. “Mother of all decidability results”
2. Muller and Schupp 1985: Configuration graphs of pushdown automata.
3. Caucal (ICALP 1996): Prefix-recognizable graphs (= ϵ-closures of configuration graphs of pushdown automata, Stirling 2000).
4. Knapik, Niwiński and Urzyczyn (TLCA 2001, FOSSACS 2002): ***PushdownTree***$_n\Sigma$ = Trees generated by order-n pushdown automata. ***SafeRecSchTree***$_n\Sigma$ = Trees generated by order-n safe recursion schemes.

Structures with decidable MSO theories: some milestones

[In timeline below, each item subsumes developments in preceding items.]

1. Rabin 1969: Regular trees. “Mother of all decidability results”
2. Muller and Schupp 1985: Configuration graphs of pushdown automata.
3. Caucal (ICALP 1996): Prefix-recognizable graphs (= ϵ-closures of configuration graphs of pushdown automata, Stirling 2000).
4. Knapik, Niwiński and Urzyczyn (TLCA 2001, FOSSACS 2002):
   PushdownTree$_n\Sigma$ = Trees generated by order-n pushdown automata.
   SafeRecSchTree$_n\Sigma$ = Trees generated by order-n safe recursion schemes.
5. Caucal (MFCS 2002). ***CaucalTree***$_n\Sigma$ and ***CaucalGraph***$_n\Sigma$.
   Theorem (KNU-C). For every $n \geq 0$,
   PushdownTree$_n\Sigma =$ ***SafeRecSchTree***$_n\Sigma =$ ***CaucalTree***$_n\Sigma$.

Structures with decidable MSO theories: some milestones

[In timeline below, each item subsumes developments in preceding items.]

1. Rabin 1969: Regular trees. "Mother of all decidability results"
2. Muller and Schupp 1985: Configuration graphs of pushdown automata.
3. Caucal (ICALP 1996): Prefix-recognizable graphs (= ϵ-closures of configuration graphs of pushdown automata, Stirling 2000).
4. Knapik, Niwiński and Urzyczyn (TLCA 2001, FOSSACS 2002):
   PushdownTree$_n\Sigma$ = Trees generated by order-n pushdown automata.
   SafeRecSchTree$_n\Sigma$ = Trees generated by order-n safe recursion schemes.
5. Caucal (MFCS 2002). ***CaucalTree***$_n\Sigma$ and ***CaucalGraph***$_n\Sigma$.
   Theorem (KNU-C). For every $n \geq 0$,
   PushdownTree$_n\Sigma =$ ***SafeRecSchTree***$_n\Sigma =$ ***CaucalTree***$_n\Sigma$.

Question. Do Σ-labelled trees generated by unsafe recursion schemes have decidable MSO theories? If so, at which orders?

Hierarchies of Finitely-Presentable Infinite Structures

Safe recursion schemes are a robust definition: several characterisations

Equivalent Higher-Order Generating Devices	**Classes of Structures** Word Languages	Trees	Graphs
Pushdown Automata	Maslov 74, 76	KNU 02	Cachat, Caucal, etc.
Safe Recursion Schemes	Damm 82	KNU 02	?
Indexed Grammars	Maslov 76	?	?

	Word Languages	*Trees*
Order 0	Regular languages	Regular trees (Rabin, etc.)
Order 1	Context-free languages; e.g. $a^n\, b^n$	Algebraic trees (Bourcelle, etc.)
Order 2	Indexed languages; e.g. $a^n\, b^n\, c^n$	Hyperalgebraic trees (KNU 01)
...	...	...

Open Problems about the Maslov (= Damm) Hierarchy

Not much is known about order-3 and above.

1. **Pumping Lemma** (or Myhill-Nerode-type results)
   There are “pumping lemmas” for orders 0, 1 and 2 ([Hay73,Gil96]).
   Pace [Blumensath04] for whole Maslov Hierarchy – runs are pumpable, conditions given as lengths of runs and configuration size.
2. **Logical Characterization.**
   Regular languages are exactly those that are MSO definable (Büchi ’60).
   There is a characterization of context-free languages using quantification over matchings [LST94].
3. **Complexity-Theoretic Characterization.**
   Engelfriet ’83, ’91: characterizations of languages accepted by alternating / two-way / multi-head / space-auxiliary order-n PDA in terms of time-complexity classes (but no result for Maslov Hierarchy itself).
4. **Relationship with Chomsky Hierachy**.
   E.g. Is order 3 context-sensitive?

What is the safety constraint?

W. Damm: Derived types in "IO and OI Hierarchies", TCS 1982.

Definition [KNU02]. An order-2 equation is **unsafe** if the RHS has a subterm P such that

1. P is order 1
2. P occurs in an operand position (i.e. as 2nd argument of the application operator)
3. P contains an order-0 parameter.

Examples of unsafe equations:

$F : (o \to o) \to o \to o \to o,\ G : o \to o,\ H : (o \to o) \to o,\ \underline{f} : o^2 \to o.$

$$\begin{array}{rcl} G\,x & = & H\,\underline{(\underline{f}\,x)} \\ F\,\varphi\,x\,y & = & f\,(F\,\underline{(F\,\varphi\,y)}\,y\,(\varphi\,x))\,\underline{a} \end{array}$$

Safety (as presented above) seems syntactically awkward and semantically unnatural but (we shall see shortly) it has important algorithmic value.

The Safe Lambda Calculus

In what sense is a safe λ-term *safe*?

A basic idea in lambda calculus / logic:

When performing β-reduction, one must use *capture-avoiding* substitution, which is standardly implemented by *renaming bound variables* afresh upon each substitution.

There is a price to pay for renaming:

Any machine that correctly computes:

$$\begin{cases} \text{INPUT:} & \text{A simply-typed } \lambda\text{-term } M \\ \text{OUTPUT:} & \text{A } \beta\text{-reduction sequence from } M \end{cases}$$

needs an *unbounded* supply of names, and hence unbounded memory.

Safety lets us get away with no renaming of bound variables!

Safety *reformulated* as a simply-typed theory

We reexpress (and generalize) the safety constraint as a simply-typed theory. Sequents have the form

$$\underbrace{x_1 : A_1, \cdots, x_i : A_i}_{\text{order } l_1} \mid \quad \cdots \quad \mid \underbrace{x_l : A_l, \cdots, x_n : A_n}_{\text{order } l_m} \vdash M : B$$

- Each A_i and B are homogeneous[1].
- Typing context partitioned according to orders with $l_1 \geq \cdots \geq l_m$.

Formation rules must respect the partition:

- When forming abstraction, all variables of the lowest type-partition must be abstracted in an atomic step.
- When forming application, the operator-term must be applied to all operand-terms (one for each type) of the highest type-partition, in one atomic step.

---

[1] o is homogeneous; and $(A_1 \to \cdots \to A_n \to o)$ is homogeneous just if $order(A_1) \geq order(A_2) \geq \cdots \geq order(A_n)$, and each A_i is homogeneous.

Safe λ-Calculus: System $\mathcal{S}$ Typing Rules

$$\frac{(\overline{A_1} \mid \cdots \mid \overline{A_n} \mid o) \text{ homogeneous} \qquad b \text{ is a type-}B \text{ constant}}{\overline{x_1} : \overline{A_1} \mid \cdots \mid \overline{x_n} : \overline{A_n} \vdash b : B}$$

$$\frac{(\overline{A_1} \mid \cdots \mid \overline{A_n} \mid o) \text{ homogeneous}}{\overline{x_1} : \overline{A_1} \mid \cdots \mid \overline{x_n} : \overline{A_n} \vdash x_{ij} : A_{ij}}$$

$$\frac{\overline{x_1} : \overline{A_1} \mid \cdots \mid \overline{x_{n+1}} : \overline{A_{n+1}} \vdash M : B \qquad (\overline{A_{n+1}} \mid B) \text{ homogeneous}}{\overline{x_1} : \overline{A_1} \mid \cdots \mid \overline{x_n} : \overline{A_n} \vdash \boldsymbol{\lambda} \overline{x_{n+1}}.M : (\overline{A_{n+1}} \mid B)}$$

$$\frac{\Gamma \vdash M : (\overline{B_1} \mid \cdots \mid \overline{B_m} \mid o) \qquad \Gamma \vdash N_1 : B_{11} \ \cdots \ \Gamma \vdash N_{l_1} : B_{1 l_1}}{\Gamma \vdash M N_1 \cdots N_{l_1} : (\overline{B_2} \mid \cdots \mid \overline{B_m} \mid o)}$$

When forming abstraction, all variables of the lowest-order type-partition must be abstracted. When forming application, the operator-term must be applied to all operand-terms (one for each type) of the highest-order type-partition.

Safe λ-calculus makes algorithmic sense

Example. Suppose $f : o^2 \rightarrow o$. Contracting the β-redex without renmaing

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

leads to variable capture. The term is *not* safe.

Theorem. "Safe λ-calculus = (a) α-conversion-free λ-calculus"
In the safe lambda calculus, there is no need to rename bound variables when performing substitution $M[N_1/\varphi_1, \cdots, N_n/\varphi_n]$ provided the substitution is performed simultaneously on all free variables of the same order in M.

Proof idea. Suppose φ free in M, and x free in N, and x captured in (capture permitting) $M[N/\varphi]$. Then M looks like $\cdots(\boldsymbol{\lambda}x.\cdots\varphi\cdots)\cdots$.
Case analysis by comparing $order(x)$ with $order(\varphi)$.

Lemma. A free variable in a safe term has order as least that of the term. $\square$

Thus when reducing a safe λ-term, we do not need any supply of fresh name.

What is the right way to think of the Safe Lambda Calculus?

Safe λ-calculus seems of independent interest, and we don't understand it.

Design issues: Is the homogeneity assumption really necessary?

Proof theory: What kind of reasoning principles does it support (via Curry-Howard)? Is it useful to automated deduction / theorem proving?

What is a model of safe λ-calculus? Does it have interesting models?

Game semantics: What kind of pointer economy does safety determine?
Ans: Pointers are redundant in safe view-functions!
E.g. Kierstead terms: $\boldsymbol{\lambda} f.f(\boldsymbol{\lambda} x.f(\boldsymbol{\lambda} y.y))$ is safe, but $\boldsymbol{\lambda} f.f(\boldsymbol{\lambda} x.f(\boldsymbol{\lambda} y.x))$ is unsafe.

Implicit complexity. Simply-typed λ-calculus characterize polytime-computable numeric functions (Leivant-Marion 93). What about the safe terms?

Nevertheless, we shall prove that safety is *not* necessary for MSO decidability.