Algebraic Operations over First and Higher-Order Terms

A Judgmental Reconstruction

Alberto Momigliano
(joint work with Frank Pfenning)

MCS Internal Seminar
November 23, 2001

1. Unification and Anti-Unification
2. The Relative Complement Problem
3. Higher-Order Patterns
4. A Strict λ-Calculus
5. The Complement and Unification Algorithm

Basics

- Let Σ be a non-trivial (at least two elements) first-order signature and $T_\Sigma(\mathcal{X})$ the set of open terms over variables $\mathcal{X}$.

- Assume a well-behaved notion of substitution (monoidal).

- Subsumption partial order: $s \leq t$ iff $\exists \theta : s \equiv \theta t$. Similarly for substitutions: $\theta \leq \theta'$ iff $\exists \phi : \theta = \theta' \cdot \phi$.

- A **unifier** for s, t is an idempotent substitution θ such that $\theta s \equiv \theta t$.

- A unifier μ for s, t is **most general** if for every other unifier θ there is a substitution ϕ such that $\theta = \mu \cdot \phi$. μt is the **most common instance** of s, t.

Computing Most Common Instances

- Let M be a **linear** term, i.e. no repeated occurrences of the same variable. Let's use a higher-order notation

- Most common instance: $M_1 \cap M_2 = M$.

$$\frac{}{x \cap M = M} \cap\_Var \qquad \frac{f \not\equiv g}{f\ \overline{M_n} \cap g\ \overline{N_m} = \perp} \cap \perp$$

$$\frac{M_1 \cap N_1 = Q_1 \cdots M_n \cap N_n = Q_n}{h\ \overline{M_m} \cap h\ \overline{N_n} = h\ \overline{Q_n}} \cap\_App$$

- Example: $f(x, b) \cap f(a, y) = f(a, b)$.

- Unification computes the greatest lower bound: if $M_1 \cap M_2 = Q$, then $Q \leq M_i$ and for every $N \leq M_i$, $N \leq Q$ holds. Proof: by rule induction.

Anti-Unification

- Let $F : T_{\Sigma}(\mathcal{X}) \times T_{\Sigma}(\mathcal{X}) \to \mathcal{X}$ be any bijection; we use it as a table to map disagreement pairs to the same variable.

- Anti-unification or least common anti-instance:

$$\frac{M_1 \cup N_1 = Q_1 \cdots M_n \cup N_n = Q_n}{h\ \overline{M_m} \cup h\ \overline{N_n} = h\ \overline{Q_n}} \cup\_App$$

$$\frac{\text{otherwise}}{N \cup M = F(N, M)} \cup Clsh$$

- Example: $f(a, a) \cup f(b, b) = f(x, x)$, where $F(a, b) = x$.

- Dually, anti-unification computes the least upper bound.

Unification for non-Linear Terms

- The linearity restriction is non-necessary, but greatly simplifies the treatment: no need to record substitutions, no need occur check.

- Non-linear unification $M_1 \cap M_2 = M, \mu$, where $\theta_{i+1} = \theta_1 \cdot \ldots \cdot \theta_i$ and $Q_i^* \equiv \theta_i Q_i$.

$$\frac{}{x \cap x = x, \epsilon}\ Flex - Same \qquad \frac{x \notin freevars(M)}{x \cap M = M, [x \leftarrow M]}\ \cap\_FlexRgd$$

$$\frac{M_1 \cap N_1 = Q_1, \theta_1 \cdots (\theta_{n-1} M_n) \cap (\theta_{n-i} N_n) = Q_n, \theta_n}{h\ \overline{M_m} \cap h\ \overline{N_n} = h\ \overline{Q_n^*}, \theta_n}\ \cap\_App$$

- Unification still computes the greatest lower bound: if $M_1 \cap M_2 = Q, \mu$, then $Q \leq M_i$ and for every $N \leq M_i$, $N \leq Q$ holds and $\mu = mgu(M_1, M_2)$.

The Lattice of Terms

- Take the set of (open) terms $T_{\Sigma}(\mathcal{X})$ of a non-trivial signature, modulo variable-renaming.

- Augment it with a constant $\perp$ to signal failure of unification.

- It is a (complete) lattice where [anti]unification is glb[lub] and the top is the equivalence class of variables.

- References:[Plotkin'70,Reynolds'70,Huet'76,Pfenning'91]

- Not a pleasant one: not distributive, not orthomodular, definitely not boolean.

Complement Problems

- An open term t as the intensional representation of its ground instances $\|t\|$.
- The *complement* of t is the set of ground terms which are not instances of t i.e. are not in $\|t\|$, where ',' represents set-theoretic union.

$$\|z_{nat}\| - \|s(0)\| \;=\; \|0, s(s(Y))\|$$

- 'Multiple Complement': by De Morgan rules:

$$\begin{aligned}
\|z_{nat}\| - \|0, s(s(Y))\| &= \{s(0)\} \\
\mathsf{Not}\{0, s(s(Y))\} &= \mathsf{Not}(0) \cap \mathsf{Not}(s(s(Y))) \\
&= s(X) \cap \{0, s(0)\} = \{s(0)\}
\end{aligned}$$

Relative Complement

- addition modulo 2:

$$\begin{array}{rcl} s(s(0)) & \longmapsto & 0 \\ y + 0 & \longmapsto & y \\ 0 + y & \longmapsto & y \\ y + y & \longmapsto & 0 \end{array}$$

- The following relative complement problem states the sufficient completeness of the rules.

$$\|x_1 + x_2\| - \|s(s(0)), 0 + y, y + 0, y + y\| = \emptyset$$

$$\exists x_1 x_2 \forall y : (z = x_1 + x_2) \wedge (z \neq s(s(0))) \wedge (z \neq y + 0) \wedge (z \neq 0 + y) \wedge (z \neq y + y)$$

- All the ground instances of a given (finite) set of terms which are not instances of another given one.

$$\|t_1, \ldots, t_n\| - \|u_1, \ldots, u_m\|$$

Applications

- Functional programming:

  1. exhaustivity and disjointness of function definition by pattern matching; production of non-ambiguous set of patterns.

  2. improved compilation: if a second clause applies, the first one does not, i.e. terms reducible by second rule are in the complement of the LHS of the first.

- Rewriting systems: *Ground reducibility* and *sufficiently completeness* of an algebraic (equational) specification. If not complete, the algorithm my lead to recover the missing cases.

Applications, II

"Stop losing sleep over incomplete specifications"

- Back to addition modulo 2. Suppose we omit $y + 0 \longmapsto y$. We can recover the missing LHS:

$$\|x_1 + x_2\| - \|s(s(0)), 0 + y, y + y\| = y + 0$$

- Infinite transition systems: the complement of a LHS returns the states from which no transition is achievable (temporal analysis of communicating processes).

- In machine learning, a concept can be captured by a term with some (finite) exceptions: its computation is the search for an explicit representation of the former.

Applications III

- inductive theorem proving : *induction-less induction* or *proof by consistency*.

- Logic Programming:

  1. Answers to negative goal as the complement of answers to its positive version [Kunen 1987].

  2. Compilation of constructive negation [Chan 1989, Stuckey 1995].

  3. The transformational approach to negation [Sato& Tamaki 1985, Barbuti et al. 1990].

First-Order Term Complement

Let Σ be a many-sorted signature and t be a linear term. Define

$$\mathsf{Not}(t : A) = \{N \mid \mathsf{Not}(t) \Rightarrow N : A\}$$

using the following rules:

no rule for variables

$$\frac{g \in \Sigma, g{:}A_1 \to \ldots \to A_m \to a \quad m \geq 0, h \not\equiv g}{\mathsf{Not}(h\ \overline{M_n}) \Rightarrow (g\ Z_1 \ldots Z_m) : a}\ \mathsf{Not}\_App^1$$

$$\frac{h \in \Sigma, h{:}A_1 \to \ldots \to A_n \to a, 1 \leq i \leq n, \vdash \mathsf{Not}(M_i) \Rightarrow N : A_i}{\mathsf{Not}(h\ \overline{M_n}) \Rightarrow (h\ Z_1 \ldots Z_{i-1}\ N\ Z_{i+1} \ldots Z_m) : a}\ \mathsf{Not}\_App^2$$

- Example: $\mathsf{Not}(s(0)) = \{0, s(s(Y))\}$.

Linearity

- **CAUTION**: the intensional representation of arbitrary terms is not closed under complementation.

- Example: let $\Sigma = a : i, f : i \to i \to i$:

  $$\|z\| - \|f(y,y)\| = \{a\} \cup \{f(s_1, s_2) \mid s_i \text{ ground}, s_1 \neq s_2\}$$

  But, incorrectly

  $$\text{Not}(f(y,y)) \stackrel{?}{=} \{a\}$$

  Other approaches:

  1. *Constrained terms*, where terms as $f(x,z) \mid x = z$ are allowed [Comon].

  2. Complement operations based on disunification [Comon, Lugiez].

The Boolean Algebra of Finite Sets of Terms

- Extends intersection (unification) and complement to finite sets of terms:

$$\mathcal{M} \cap \mathcal{N} = \bigcup_{M \in \mathcal{M}, N \in \mathcal{N}} M \cap N$$

$$\mathsf{Not}(\mathcal{M}) = \bigcap_{M \in \mathcal{M}} \mathsf{Not}(M)$$

- Let union be set-theoretic (anti-unification won't work).

- Add bottom and let top the singleton containing a variable.

- It is a boolean algebra under extensional equality, i.e.

$$\mathcal{N}_1 \simeq \mathcal{N}_2 \ \text{ iff } \ \|\mathcal{N}_1\| = \|\mathcal{N}_2\|$$

Higher-Order Patterns over Simple Types

$$
\begin{array}{rrcl}
\textit{Simple Types} & A & ::= & a \mid A_1 \to A_2 \\
\textit{Terms} & M & ::= & c \mid x \mid \lambda x{:}A.M \mid M_1\, M_2 \\
\textit{Signatures} & \Sigma & ::= & \cdot \mid \Sigma, P{:}\mathsf{type} \mid \Sigma, c{:}A \\
\textit{Contexts} & \Gamma & ::= & \cdot \mid \Gamma, x{:}A
\end{array}
$$

- Algebraic operations require *canonical forms*, that is η-long β-normal forms.

- Patterns: add unification variables $E, F \ldots$ and require them to be applied to distinct bound variables. Examples: $\lambda x.\, c(E\ x)$, $\lambda x \lambda y\, .\, F\ y\ x$. Non-Examples: $\lambda x.\, E\ c$, $\lambda x.\, F(E\ x)$, $\lambda x.\, E\ x\ x$.

$$
\textit{Patterns} \quad M \quad ::= \quad \lambda x{:}A.M \mid h\ M_1 \ldots M_n \mid E_A\ x_1 \ldots x_n
$$

Pattern Unification

Let M be a linear term, Φ_i an array of distinct bound variables:

$$\frac{\Phi_1 \cap \Phi_2 = \Phi}{\Gamma \vdash E\ \Phi_1 \cap F\ \Phi_2 = H\ \Phi : a} \cap\_FlexFlex$$

no rule for flex-flex same

$$\frac{}{\Gamma \vdash E\ \Phi \cap h\ \overline{M_n} = h\ \overline{M_n} : a} \cap\_FlexRgd$$

$$\frac{\Gamma, x{:}A \vdash M \cap N = Q : B}{\Gamma \vdash \lambda x : A.\, M \cap \lambda x : A.\, N = \lambda x : A.\, Q : A \to B} \cap\_Lam$$

$$\frac{h \in \Gamma \cup \Sigma \quad \Gamma \vdash M_1 \cap N_1 = Q_1 : A_1 \cdots \Gamma \vdash M_n \cap N_n = Q_n : A_n}{\Gamma \vdash h\ \overline{M_m} \cap h\ \overline{N_n} = h\ \overline{Q_n} : a} \cap\_App$$

Example:

$(\lambda x.\, h\ (E\ x)\ x\ F) \cap (\lambda x.\, h\ 0\ (E'\ x)(F'\ x)) = (\lambda x.\, h\ 0\ x\ H)$.

Pattern Complement

- **Note**: the (full) second-order relative complement problem is not semi-decidable. Higher-order disunification (which encompasses complement problems) on higher-order patterns is decidable [Lugiez 1995].

- Secondly, the treatment differentiates depending on whether the pattern is *fully applied*, that is a unification variable is applied to **every** bound variable mentioned in the lambda binder, Example $\lambda x.\, \lambda y.\, (E\ y\ x)$, non-example $\lambda x.\, \lambda y.\, h\ (E\ x)$

- For fully applied patterns proceed similarly to first-order with an additional case for bound variables, collected in a context Γ, so that the complement is taken w.r.t. both the signature and the current context.

Fully Applied Pattern Complement

Let $h, g \in \Sigma \cup \Gamma$. For a linear M such that $\Gamma \vdash M : A$ define
$\mathsf{Not}(M : A) = \{N \mid \cdot \vdash \mathsf{Not}(M) = N : A\}$

$$\frac{\Gamma, x{:}A \vdash \mathsf{Not}(M) \Rightarrow N : B}{\Gamma \vdash \mathsf{Not}(\lambda x : A.\, M) \Rightarrow \lambda x : A.\, N : A \to B}\ \mathsf{Not}Lam$$

$$\frac{\Gamma \vdash g{:}A_1 \to \ldots \to A_m \to a \quad m \geq 0, h \not\equiv g}{\Gamma \vdash \mathsf{Not}(h\ \overline{M_n}) \Rightarrow (\ldots(g\ (Z_1\Gamma))\ldots(Z_m\Gamma)) : a}\ \mathsf{Not}A^1$$

$$\frac{\Gamma \vdash h{:}A_1 \to \ldots \to A_n \to a, 1 \leq i \leq n, \Gamma \vdash \mathsf{Not}(M_i) \Rightarrow N : A_i}{\Gamma \vdash \mathsf{Not}(h\ \overline{M_n}^1) \Rightarrow (\ldots(h\ (Z_1\Gamma))\ldots(Z_{i-1}\Gamma)\ N\ (Z_{i+1}\Gamma)\ldots(Z_m\Gamma)) : a}\ \mathsf{Not}A^2$$

where $m \geq 0$ and $(Z\ \Gamma)$'s denotes the application of a fresh variable of appropriate typing to the domain of the context.

Encoding the (untyped) λ-calculus

$$
\begin{array}{lcl}
e & ::= & x \mid (\boldsymbol{\lambda} x \,.\, e) \mid e_1 \; e_2 \\
\Sigma & ::= & \mathit{exp} : \text{type}, \mathit{lam} : (\mathit{exp} \to \mathit{exp}) \to \mathit{exp}, \mathit{app} : \mathit{exp} \to \mathit{exp} \to \mathit{exp}
\end{array}
$$

- The representation function:

  $$\ulcorner x \urcorner = x \qquad \ulcorner \boldsymbol{\lambda} x \,.\, e \urcorner = \mathit{lam}\ (\lambda x : \mathit{exp}. \ulcorner e \urcorner) \qquad \ulcorner e_1 \; e_2 \urcorner = \mathit{app}\ \ulcorner e_1 \urcorner \ulcorner e_2 \urcorner$$

- The encoding a β-redex as a pattern:

  $$\ulcorner (\boldsymbol{\lambda} x \,.\, e \; x) \; f \urcorner \;=\; \mathit{app}\ (\mathit{lam}\ (\lambda x : \mathit{exp}. (E\ x))\ F)$$

- Complement of the β-redex in the empty context is $\boldsymbol{\lambda} x \,.\, e, (e_1 \; e_2) \; e_3$, that is

  $$\begin{array}{l} \cdot \vdash \text{Not}(\mathit{app}\ (\mathit{lam}\ (\lambda x : \mathit{exp}. (E\ x))\ F) = \\ \qquad \{\mathit{lam}\ (\lambda x : \mathit{exp}. H\ x), \mathit{app}\ (\mathit{app}\ H_1\ H_2)\ H_3\} \end{array}$$

Encoding the λ-calculus: Partially Applied Patterns 2/2

- Encoding a η-redex, where $x \notin FV(e)$:

$$\ulcorner \boldsymbol{\lambda} x\,.\,(e\ x) \urcorner \;=\; lam(\lambda x\!:\!exp.\,(\boxed{app\ E})\ x)$$

- Hence e *does* **not** depend on x. Intuitively its complement should contain those application sub-terms where E **does** depend on x:

$$lam(\lambda x\!:\!exp.\,(\boxed{app\ F\#x})\ (H\ x))$$

- The simply typed λ-calculus theory is not strong enough to directly represent the complement of non-fully applied patterns.

- This cannot be avoided similarly to left-linearization. We need to have a notion of *strictness* in the language.

Strictness

- Lugiez modifies the language of terms to promote constraints to first-class objects, For example $\lambda xyz.\, M[1,3]$ would denote a function which depends on its first and third argument. The technical handling of those objects then becomes awkward.

- We generalize our language to include

  1. *strict functions* of type $A \stackrel{1}{\to} B$ (which are guaranteed to depend on their argument)

  2. *invariant functions* of type $A \stackrel{0}{\to} B$ (which are guaranteed **not** to depend on their argument).

  3. The full function space $A \stackrel{u}{\to} B$.

The language of strict types

- Any concretely given function either will or will not depend on its argument, but in the presence of unification variables we still need the ability to remain uncommitted.

$$
\begin{array}{rrcl}
\textit{Labels} & k & ::= & 1 \mid 0 \mid u \\
\textit{Types} & A & ::= & o \mid P \mid A_1 \xrightarrow{k} A_2 \\
\textit{Terms} & M & ::= & c \mid x \mid \lambda x^k{:}A.M \mid M_1\, M_2^k \\
\textit{Contexts} & \Gamma & ::= & \cdot \mid \Gamma, x{:}A
\end{array}
$$

- Typing judgment: $\Gamma; \Omega; \Delta \vdash M : A$. Compare with the two-context presentation of linear logic: $\Gamma; \Delta \vdash M : A$.

- Logically speaking, this lives in the galaxy of relevance logics.

Typing rules

$$\frac{c{:}A \in \Sigma}{\Gamma; \Omega; \cdot \vdash c : A}\,Con \qquad \textit{no } Id^0 \textit{ rule}$$

$$\frac{}{(\Gamma, x{:}A); \Omega; \cdot \vdash x : A}\,Id^u \qquad \frac{}{\Gamma; \Omega; x{:}A \vdash x : A}\,Id^1$$

$$\frac{(\Gamma, x{:}A); \Omega; \Delta \vdash M : B}{\Gamma; \Omega; \Delta \vdash \lambda x^u{:}A.M : A \xrightarrow{u} B}\,\xrightarrow{u} I$$

$$\frac{\Gamma; (\Omega, x{:}A); \Delta \vdash M : B}{\Gamma; \Omega; \Delta \vdash \lambda x^0{:}A.M : A \xrightarrow{0} B}\,\xrightarrow{0} I$$

$$\frac{\Gamma; \Omega; (\Delta, x{:}A) \vdash M : B}{\Gamma; \Omega; \Delta \vdash \lambda x^1{:}A.M : A \xrightarrow{1} B}\,\xrightarrow{1} I$$

Typing rules

$$\frac{\Gamma; \Omega; \Delta \vdash M : A \xrightarrow{u} B \qquad (\Gamma, \Delta); \Omega; \cdot \vdash N : A}{\Gamma; \Omega; \Delta \vdash M\, N^u : B} \xrightarrow{u} E$$

$$\frac{\Gamma; \Omega; \Delta \vdash M : A \xrightarrow{0} B \qquad (\Gamma, \Omega, \Delta); \cdot; \cdot \vdash N : A}{\Gamma; \Omega; \Delta \vdash M\, N^0 : B} \xrightarrow{0} E$$

$$\frac{(\Gamma, \Delta_N); \Omega; \Delta_M \vdash M : A \xrightarrow{1} B \qquad (\Gamma, \Delta_M); \Omega; \Delta_N \vdash N : A}{\Gamma; \Omega; (\Delta_M, \Delta_N) \vdash M\, N^1 : B} \xrightarrow{1} E$$

$$\dfrac{\dfrac{\dfrac{}{y; \cdot; x \vdash x : A \xrightarrow{1} A \xrightarrow{1} B}\,\mathtt{Id}^1 \quad \dfrac{}{x; \cdot; y \vdash y : A}\,\mathtt{Id}^1}{\cdot; \cdot; (x, y) \vdash x\ y^1 : A \xrightarrow{1} B} \xrightarrow{1} E \qquad \dfrac{}{(x, y); \cdot; \cdot \vdash y : A}\,\mathtt{Id}^u}{\cdot; \cdot; (x{:}A \xrightarrow{1} A \xrightarrow{1} B, y{:}A) \vdash (x\ y^1)\ y^1 : B} \xrightarrow{1} E$$

Complementing Patterns in the Strict Calculus

- How we embed the original λ-calculus, and is the calculus now closed under complement?

- Example:

$$\ulcorner \lambda x.\, \lambda y.\, x \urcorner = lam\ (\lambda x\!:\!exp.\, lam\ (\lambda y\!:\!exp.\, x))$$

- Argument to the first occurrence of 'lam' is strict, argument to the second is invariant. Type must be $(exp \stackrel{u}{\rightarrow} exp) \stackrel{1}{\rightarrow} exp$.

- Idea: make the status of terms as determined as possible, while preserving the possibility of well-typing sub-terms as 'lam' above.

Simple Terms

- Constants (and bound variables) are strict functions, higher order functions (negative occurrences) must be undetermined.

- Example:

  $lam\ (\lambda x\!:\!exp.\,lam\ (\lambda y\!:\!exp.\,x)) \mapsto lam\ (\lambda x^u\!:\!exp.\,lam\ (\lambda y^u\!:\!exp.\,x)^1)^1$

- Bring to full application:

  $lam(\lambda x^y\!:\!exp.\,(app\ \boxed{E})^1\ x^1) \mapsto lam(\lambda x^u\!:\!exp.\,(app\ (\boxed{E\ x^0})^1\ x^1))$

- The resulting language:

Simple Terms $M\ ::=\ \lambda x^u{:}A.M \mid (\ldots(h\ M_1)^1 \ldots M_n)^1 \mid E_A\ \overline{x_n}^l$

The Complement Algorithm

- For logic variables consider each argument in turn:

  1. If an argument variable is **undetermined** it does not contribute to the negation.

  2. If an argument variable is **strict**, complement the corresponding label from 1 to 0 while all other arguments are undetermined.

  3. For **vacuous** argument variables proceed dually.

- Example:

$$\begin{aligned}\text{Not}(E\ x^u y^1) &= \{F\ x^u y^0\} \\ \text{Not}(E\ x^0 y^1) &= \{F\ x^1 y^u, G\ x^u y^0\}\end{aligned}$$

Complement of Not($lam(\lambda x^u\!:\!exp.\, app\ (E\ x^0)\ x)$)

different from first coordinate $\boxed{E\ x^0}$

$\{lam(\lambda x^u\!:\!exp.\, app\ (\boxed{H\ x^1})\ (H'\ x^u)),$

different from second coordinate $\boxed{x}$

$lam(\lambda x^u\!:\!exp.\, app\ (H\ x^u)\ \boxed{(app\ (H'\ x^u)\ (H'' x^u))})$

$lam(\lambda x^u\!:\!exp.\, app\ (H\ x^u)\ \boxed{lam(\lambda y^u\!:\!exp.\, H'\ x^u\ y^u)}),$

different from $\boxed{(app\ (E\ x^0)\ x)}$

$lam(\lambda x^u\!:\!exp.\boxed{lam(\lambda y^u\!:\!exp.\, H\ x^u\ y^u)}),$

$lam(\lambda x^u\!:\!exp.\boxed{x}),$

different from top $\boxed{lam(\ldots)}$ functor

$\boxed{app\ H\ H'}\}$

Unification of Simple Terms

- Unification as *intersection* of labels, (analogous to conjunction in three-valued logic).

- Unification is not unary, but *finite* sets of most general unifiers exist.

- Example: since x is strict in the LHS, it can be strict either in the first or second subterm of c:

$$x{:}A \vdash E\ x^1 \cap\ c\ (F\ x^u)^1\ (G\ x^u)^1 = \\ \{c\ (G\ x^1)^1\ (H\ x^u)^1, c\ (G\ x^u)^1\ (H\ x^1)^1\}$$

- Rules: skipped.

- Anti-unification: open problem (but is it interesting?).

Results

- **Theorem** If M is a simple term, so is Not(M).

- **Theorem** M is a ground instance of Not(N) if and only if M is **not** a ground instance of N.

- **Theorem** M is a ground instance of N_1 and N_2 if and only if M is a ground instance of $N_1 \cap N_2$.

- **Corollary** *Finite* sets of simple terms form a boolean algebra, under unification, set-theoretic union and complement.

- Define relative complement as $\mathcal{M} - \mathcal{N} = \mathcal{M} \cap \mathrm{Not}(\mathcal{N})$.

Towards Intensional Negation

$$
\begin{array}{lll}
betarx & : & isredx(app\ (lam\ (\lambda x^u : exp.\, E\ x^u))\ F) \\
etarx & : & isredx(lam(\lambda x^u : exp.\, app\ (E\ x^0)\ x))
\end{array}
$$

Infer its negation by computing:

$\mathsf{Not}(app\ (lam\ (\lambda x^u : exp.\, E\ x^u))\ F) \cap \mathsf{Not}(lam(\lambda x^u : exp.\, app\ (M\ x^0)\ x))$

$$
\begin{array}{ll}
nb1: & non\_isredx(lam(\lambda x^u : exp.\, app\ (H\ x^1)\ (H'\ x^u))) \\
nb2: & non\_isredx(lam(\lambda x^u : exp.\, app\ (H\ x^u)\ (app\ H'\ x^u H''\ x^u))) \\
nb3: & non\_isredx(lam(\lambda x^u : exp.\, app\ (H\ x^u)\ (lam(\lambda y^u : exp.\, H'\ x^u\ y^u)))) \\
nb4: & non\_isredx(lam(\lambda x^u : exp.\, lam(\lambda y^u : exp.\, H\ x^u\ y^u))) \\
nb5: & non\_isredx(lam(\lambda x^u : exp.\, x)) \\
nb6: & non\_isredx(app\ (app\ H_1\ H_2)\ H_3\ )
\end{array}
$$

Conclusions

- The (relative) complement problem is interesting
- Even restricted to patterns, it does not generalize immediately to the higher-order case: we need to have an intrinsic notion of strictness
- We have presented a strict λ- calculus, which we think gives a contribution to the understanding of sub-structural calculi
- We have shown how to perform any boolean operation on **finite** sets of simple terms.

Future work

- Generalize the complement algorithm beyond simple terms, allowing arbitrary abstractions, though complement on **any** strict term is not quite possible.

- Revisit higher-order pattern disunification exploiting strict types.

- Extend to the intuitionistic and linear dependently-typed case (ongoing).

All items are non-trivial.