The Relative Complement Problem for Higher-Order Patterns

Alberto Momigliano
Carnegie Mellon University

joint work with Frank Pfenning
September 9 1999

Overview:

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

Complement Problem

- An open term t as the intensional representation of its ground instances, denoted $\|t\|$.
- The *complement* of t, say $\mathsf{Not}(t)$, is the set of all ground terms which are not instances of t i.e. are not in $\|t\|$.
- Example: take the signature of numerals:

$$\mathsf{Not}(s(0)) = \{0, s(s(Y))\}$$

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

- 'Multiple Complement':

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

Relative Complement

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

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

- Example: integer addition modulo 2.

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

- A relative complement problem (sufficient completeness):

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

Applications

- Functional programming:

  1. Determination of exhaustivity and disjointness of a function definition by patterns.

  2. Improved compilation by analysis of pattern applicability.

- Rewriting systems:

  1. *Sufficient completeness* of an algebraic (equational) specification.

  2. Infinite transition systems (temporal analysis of communicating processes).

Applications II

- Machine learning: a concept as a term with some (finite) exceptions.

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

- Logic Programming

  1. Answers to negative goals as complement of answers to positive ones [Kunen 1987].

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

Our Aim: extend the latter to Logical Frameworks.

First-Order Term Complement

- Consider a finite (untyped) signature Σ and a term t:

$$
\begin{aligned}
\mathsf{Not}_\Sigma(x) &= \emptyset \\
\mathsf{Not}_\Sigma(f(\overline{t_n})) &= \\
&\{g(\vec{x}) \mid \text{for all } g \in \Sigma \text{ distinct from } f\} \cup \\
&\{f(z_1, \ldots, z_{i-1}, s, z_{i+1}, \ldots, z_n) \mid \\
&s \in \mathsf{Not}_\Sigma(t_i), i \in [1, n]\}
\end{aligned}
$$

- The 'Not' function tends to produce irrelevant outcomes; $\mathsf{Not}_{ListN}(0)$ in the signature of lists of numerals contains not only $s(X)$ but also nil and $(cons\ X\ Xs)$.

- Complementation must be typed.

Linearity

- **CAUTION**: the intensional representation of terms is not closed under complementation. Let $\Sigma = a : i, f : i \to i \to i$:

$$\|z\| - \|f(y,y)\| \;\; = \;\; \{a\} \cup \{f(x,z) \mid x \neq z\}$$

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

- Correct only for *linear* terms, i.e. terms without repeated occurrence of a free variable [Lassez & Marriot 1987].

- Left-linearization [Plaisted, Stickel].

- *Constrained terms*, i.e. $f(x,z) \mid x \neq z$.

- Complement operations based on disunification [Comon].

Higher-Order Patterns over Simple Types.

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

- **Note:** the second-order relative complement problem is not semi-decidable, but disunification on higher-order patterns is decidable [Lugiez 1995].

- Patterns [Miller 1991]: terms where every free variable is applied to distinct bound variables.

  Examples: $\lambda x.\, c(E\ x)$, $\lambda x \lambda y\, .\, F\ y\ x$. Non-Examples: $\lambda x.\, E\ c$, $\lambda x.\, F(E\ x)$, $\lambda x.\, E\ x\ x$.

Fully Applied Pattern

- Complementation differentiates depending on whether the term is *fully applied*, i.e. when all variables appearing in the binder are mentioned in the scope of a free variable:
  - Example: $\lambda x \lambda y \lambda z \, . \, c(E(y, x, z))$
  - Non-Example: $\lambda x \lambda y \lambda z \, . \, E \; y$
- For fully applied terms proceed similarly to first-order with an additional case for bound variables: the complement is taken w.r.t. both the signature and the current context.
- In the non-fully applied case the complement has to take into account the occurrence issue.

Encoding the the (untyped) λ-calculus

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

The representation function:

$$\begin{array}{rcl} \ulcorner x \urcorner & = & x \\ \ulcorner \lambda x.\, e \urcorner & = & lam\ (\lambda x : exp.\, \ulcorner e \urcorner) \\ \ulcorner e_1\ e_2 \urcorner & = & app\ \ulcorner e_1 \urcorner\, \ulcorner e_2 \urcorner \end{array}$$

Encoding a β-redex:

$$\ulcorner (\lambda x.\, e\ x)\ f \urcorner \ = \ app\ (lam\ (\lambda x : exp.\, (E\ x))\ F$$

Complement of the latter:

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

FAHP Complement

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

$$\frac{g \in \Sigma \cup \Gamma, g{:}A_1 \rightarrow \ldots \rightarrow A_m \rightarrow P \quad m \geq 0, h \not\equiv g}{\Gamma \vdash \mathsf{Not}(h\ \overline{M_n}) = (\ldots(g\ (Z_1\Gamma))\ldots(Z_m\Gamma)) : P}$$

$$\frac{h \in \Sigma \cup \Gamma, \Gamma \vdash h{:}A_1 \rightarrow \ldots \rightarrow A_n \rightarrow P, 1 \leq i \leq n, \Gamma \vdash \mathsf{Not}(M_i) = N : A_i}{\Gamma \vdash \mathsf{Not}(h\ \overline{M_n}^1) = (\ldots(h\ (Z_1\Gamma))\ldots(Z_{i-1}\Gamma)\ N\ (Z_{i+1}\Gamma)\ldots(Z_m\Gamma)) : P}$$

$(Z\ \Gamma)$'s denotes the application of a fresh variable of correct typing to a context Γ.

$$\mathsf{Not}(M : A) = \{N \mid \cdot \vdash \mathsf{Not}(M) = N : A\}$$

NFA Patterns

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

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

- Hence E **does not** depend on x and the term is *not* fully applied. Its complement should contain those applications where E **does** depend on x.

- The simply typed λ-calculus is not strong enough to directly represent the complement of non-fully applied patterns expressed as a finite set of 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 (awkward).

- We *internalize* strictness in the type-system.

- We generalize our language to include:

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

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

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

The Language of Strict Types

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

- Three form of abstractions, given the label on bound variable.

- Three form of applications, as a postfix operation.

- The calculus is well-behaved (unique canonical forms exist).

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, continued

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

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

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

Complement Desiderata

- (Disjointness) It is not the case that M is both a ground instance of N and of Not(N).
- (Exhaustivity) M is a ground instance of N or M is a ground instance of Not(N).
- How we embed the original λ-calculus, and is the calculus now closed under complement?
- Exhaustivity does not hold in general as we allow logic variables in NFA terms.
- Restrict to a class of terms (that we call *simple*) for which exhaustivity holds and bring to full application those terms.

Embedding the simply-typed λ-Calculus

- 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 can only be $(exp \stackrel{u}{\to} exp) \stackrel{1}{\to} exp$.

- Constants (and bound variables) are strict functions.

- Second order functions must be undetermined.

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

Simple Terms

- In general:

  1. positive occurrence of function types are translated to strict functions.

  2. negative occurrences to undetermined functions.

- Make NFA terms fully applied:

$$\lambda x \lambda y \lambda z\,.\,E\ y \;\;\Rightarrow\;\; \lambda x^u \lambda y^u \lambda z^u\,.\,E\ x^0\ y^u\ z^0$$

- The resulting language:

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

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 we proceed dually.

$$\frac{1 \leq j \leq n \quad k \in \{1, 0\}}{\mathsf{Not}(E\ x_1^{l_1} \ldots x_{j-1}^{l_{j-1}}\ x_j^k\ x_{j+1}^{l_{j+1}} \ldots x_n^{l_n}) = Z\ x_1^u \ldots x_{j-1}^u\ x_j^{\mathsf{Not}(k)}\ x_{j+1}^u \ldots x_n^u : P}$$

no rule for $k = u$

The Complement Algorithm II

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

$$\frac{g \in \Sigma \cup \Gamma, g{:}A_1 \stackrel{1}{\to} \ldots \stackrel{1}{\to} A_m \stackrel{1}{\to} P \quad m \geq 0, h \not\equiv g}{\Gamma \vdash \mathsf{Not}(h\ \overline{M_n}^1) = (\ldots(g\ (Z_1\Gamma^u))^1 \ldots (Z_m\Gamma^u))^1 : P}$$

$$\frac{h \in \Sigma \cup \Gamma, \Gamma \vdash h{:}A_1 \stackrel{1}{\to} \ldots \stackrel{1}{\to} A_n \stackrel{1}{\to} P, 1 \leq i \leq n, \Gamma \vdash \mathsf{Not}(M_i) = N : A_i}{\Gamma \vdash \mathsf{Not}(h\ \overline{M_n}^1) = (h\ (Z_1\Gamma^u))^1 \ldots (Z_{i-1}\Gamma^u)\ N^1\ (Z_{i+1}\Gamma^u)^1 \ldots (Z_m\Gamma^u))^1 : P}$$

$$\mathsf{Not}(M : A) = \{N \mid \cdot \vdash \mathsf{Not}(M) = N : A\}$$

Examples of Complementation

If E is a variable, it has the canonical form $\lambda\overline{x_n^u}.E\ \overline{x_n^u}$: hence

$$\mathsf{Not}(E) = \emptyset$$

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

$$\begin{array}{l}\mathsf{Not}(lam(\lambda x^u{:}exp.\,app\ (E\ x^0)\ x^u)) = \\ \quad \{lam(\lambda x^u{:}exp.\,app\ (H\ x^1)\ (H'\ x^u)), \\ \quad lam(\lambda x^u{:}exp.\,app\ (H\ x^u)\ (app\ (H'\ x^u)\ (H'' x^u)), \\ \quad lam(\lambda x^u{:}exp.\,app\ (H\ x^u)\ (lam(\lambda y^u{:}exp.\,H'\ x^u\ y^u)), \\ \quad lam(\lambda x^u{:}exp.\,lam(\lambda y^u{:}exp.\,H\ x^u\ y^u)), \\ \quad lam(\lambda x^u{:}exp.\,x), \\ \quad app\ H\ H'\}\end{array}$$

Results

- **Theorem** If M is a simple term, so is N, for all $N \in \mathsf{Not}(M)$.

- **Theorem** It is not the case that both $\Gamma \vdash M \in \|N\| : A$ and $\Gamma \vdash M \in \|\mathsf{Not}(N)\| : A$.

- **Lemma** Let M be a simple patterns:

  If $(\Gamma, x{:}C); \cdot; \cdot \vdash M \Updownarrow A$ then $\Gamma; \cdot; x{:}C \vdash M \Updownarrow A$ or $\Gamma; x{:}C; \cdot \vdash M \Updownarrow A$.

- **Theorem**

  if $\mathcal{D} :: \Gamma \vdash M : A$ then $\Gamma \vdash M \in \|N\| : A$ or $\Gamma \vdash M \in \|\mathsf{Not}(N)\| : A$.

Elimination of negation in a logical framework

- Consider the following 'program':

$$isredx(app\ (lam\ (\lambda x^u\!:\!exp.\, E\ x^u))\ F)$$

$$isredx(lam(\lambda x^u\!:\!exp.\, app\ (E\ x^0)\ x^u)$$

- What about its negation? We can compute:

  $\|Z_{exp}\| - \|app\ (lam\ (\lambda x^u\!:\!exp.\, E\ x^u))\ F, lam(\lambda x^u\!:\!exp.\, app\ (M\ x^0)\ x\|$

  by *intersecting* Not$(app\ (lam\ (\lambda x^u\!:\!exp.\, E\ x^u))\ F)$ and Not$(lam(\lambda x^u\!:\!exp.\, app\ (M\ x^0)\ x^u))$

- We can implement intersection as *unification* of simple terms (not detailed here).

Elimination of negation in a logical framework, II

$non\_isredx(lam(\lambda x^u\!:\!exp.\, app\ (H\ \ x^1)\ \ H'\ \ x^u))$

$non\_isredx(lam(\lambda x^u\!:\!exp.\, app\ (H\ \ x^u)\ \ (app\ \ H'\ \ x^u H''\ \ x^u)))$

$non\_isredx(lam(\lambda x^u\!:\!exp.\, app\ (H\ \ x^u)\ \ (lam(\lambda y^u\!:\!exp.\, H'\ \ x^u\ \ y^u))))$

$non\_isredx(lam(\lambda x^u\!:\!exp.\, lam(\lambda y^u\!:\!exp.\, H\ \ x^u\ \ y^u)))$

$non\_isredx(lam(\lambda x^u\!:\!exp.\, x))$

$non\_isredx(app\ (app\ \ H_1\ \ H_2)\ \ H_3\ )$

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.

- It is possible to perform any boolean operation on sets of simple terms.

Future work

- Combine term complementation with clause complementation in higher-order hereditary formulae.

- Address the issue of complement and intersection on *any* strict term.

- Extend to the intuitionistic and linear dependent type case.