An example of negation elimination in hereditary Harrop formulae

Alberto Momigliano

Consider the language of the untyped λ-calculus:

$$e \quad ::= \quad x \mid \Lambda x \,.\, e \mid e_1 \, e_2$$

We encode these expressions as terms in (labeled) hereditary Harrop formulae via the usual techniques of higher-order abstract syntax as canonical forms over the following signature:

$$\Sigma \quad = \quad exp : \text{type}, lam : (exp \to exp) \to exp, app : exp \to (exp \to exp)$$

The representation function is given by:

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

A term is *linear* if every functional subterm uses each of its arguments exactly once: in particular, we make sure that a function is linear in its first argument and then recurring on its body.

$$\begin{array}{rcl} linlam & : & linear(lam\ \lambda x \,.\, E\ x) \\ & & \quad \leftarrow linx(\lambda x \,.\, E\ x) \\ & & \quad \leftarrow (\forall x : exp. linear(x) \to linear(E\ x)). \\ linapp & : & linear(app\ E_1\ E_2) \\ & & \quad \leftarrow linear(E_1) \\ & & \quad \leftarrow linear(E_2). \end{array}$$

$$\begin{array}{rcl} linxx & : & linx(\lambda x \,.\, x). \\ linxap1 & : & linx(\lambda x \,.\, app\ (E_1\ x)\ E_2) \\ & & \quad \leftarrow linx(\lambda x \,.\, E_1\ x). \\ linxap2 & : & linx(\lambda x \,.\, app\ E_1\ (E_2\ x)) \\ & & \quad \leftarrow linx(\lambda x \,.\, E_2\ x). \\ linxlm & : & linx(\lambda x \,.\, lam(\lambda y \,.\, E\ x\ y)) \\ & & \quad \leftarrow (\forall y : exp. linx(\lambda x \,.\, E\ x\ y)). \end{array}$$

This is clearly a decision procedure, which can be complemented; an expression is *not* linear if there is some function which either does not use its argument or uses them more than once. The following is obtained via the clause complementation algorithm described in my thesis; note that '$\neg$' is *not* a connective but a name constructor for atoms.

$$
\begin{array}{rcl}
\neg linlam1 & : & \neg linear(lam\ \lambda x\,.\,E\ x) \\
& & \quad \leftarrow \neg linx(\lambda x\,.\,E\ x). \\
\neg linlam2 & : & \neg linear(lam\ \lambda x\,.\,E\ x) \\
& & \quad \leftarrow (\forall x\!:\!exp.\,\neg linx(\lambda x\,.\,y) \wedge linear(x) \rightarrow \neg linear(E\ x)). \\
\neg linapp1 & : & \neg linear(app\ E_1\ E_2) \\
& & \quad \leftarrow \neg linear(E_1). \\
\neg linapp2 & : & \neg linear(app\ E_1\ E_2) \\
& & \quad \leftarrow \neg linear(E_2).
\end{array}
$$

$$
\begin{array}{rcl}
\neg linxap0 & : & \neg linx(\lambda x\,.\,app\ (E_1\ x^1)\ (E_2\ x^1)). \\
\neg linxap1 & : & \neg linx(\lambda x\,.\,app\ (E_1\ x)\ (E_2\ x^1)) \\
& & \quad \leftarrow \neg linx(\lambda x\,.\,E_1\ x). \\
\neg linxap2 & : & \neg linx(\lambda x\,.\,app\ (E_1\ x^1)\ (E_2\ x)) \\
& & \quad \leftarrow \neg linx(\lambda x\,.\,E_2\ x). \\
\neg linxap3 & : & \neg linx(\lambda x\,.\,app\ (E_1\ x)\ (E_2\ x)) \\
& & \quad \leftarrow \neg linx(\lambda x\,.\,E_1\ x) \wedge \neg linx(\lambda x\,.\,E_2\ x). \\
\neg linxlm & : & \neg linx(\lambda x\,.\,lam(\lambda y\,.\,E\ x\ y)) \\
& & \quad \leftarrow (\forall y\!:\!exp.\,\neg linx(\lambda x\,.\,y) \rightarrow \neg linx(\lambda x\,.\,E\ x\ y)).
\end{array}
$$

where $(E_1\ x^1)$ denotes that E_1 *must* depend on x, i.e. it strict in x; the semantics of this annotation is given by the strict λ-calculus, again described in my dissertation.