A Step-to-Step Example of Negation Elimination in Hereditary Harrop Formulae

Alberto Momigliano

University of Leicester
amomigliano@mcs.le.ac.uk

This is a detailed version of the main example in the companion paper [1]. Consider the language of the untyped Λ-calculus[1]:

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

A Λ-term is *linear* if every functional sub-term uses each of its arguments exactly once: in particular, we make sure that a function is linear in its first argument (*linear in* x) and then recur on the rest of the arguments. This can be realized by the following rules.

$$\dfrac{\Lambda x \,.\, e \text{ linear in } x \qquad \begin{array}{c} \dfrac{}{x \text{ linear}}\,u \\ \vdots \\ e \text{ linear} \end{array}}{\Lambda x \,.\, e \text{ linear}}\, \mathit{linlam}^{x,u}$$

$$\dfrac{e_1 \text{ linear} \qquad e_2 \text{ linear}}{(e_1\ e_2) \text{ linear}}\, \mathit{linapp}$$

$$\dfrac{}{\Lambda x \,.\, x \text{ linear in } x}\, \mathit{linxx} \qquad \dfrac{\Lambda x \,.\, e \text{ linear in } x}{\Lambda x \,.\, \Lambda y \,.\, e \text{ linear in } x}\, \mathit{linxlm}^{y}$$

$$\dfrac{\Lambda x \,.\, e_1 \text{ linear in } x}{\Lambda x \,.\, (e_1\ e_2) \text{ linear in } x}\, \mathit{linxapp}_1 \qquad \dfrac{\Lambda x \,.\, e_2 \text{ linear in } x}{\Lambda x \,.\, (e_1\ e_2) \text{ linear in } x}\, \mathit{linxapp}_2$$

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

$$\begin{aligned} \mathit{exp} &: \text{type}. \\ \mathit{lam} &: (\mathit{exp} \to \mathit{exp}) \to \mathit{exp}. \\ \mathit{app} &: \mathit{exp} \to (\mathit{exp} \to \mathit{exp}). \\ \mathit{linear} &: \mathit{exp} \to \mathbf{o}. \\ \mathit{linx} &: (\mathit{exp} \to \mathit{exp}) \to \mathbf{o}. \end{aligned}$$

The representation function is given by:

$$\begin{aligned} \ulcorner x \urcorner &= x : \mathit{exp} \\ \ulcorner \Lambda x \,.\, e \urcorner &= \mathit{lam}\ (\lambda x{:}\mathit{exp}. \ulcorner e \urcorner) \\ \ulcorner e_1\ e_2 \urcorner &= \mathit{app}\, \ulcorner e_1 \urcorner\, \ulcorner e_2 \urcorner \\ \ulcorner e \text{ linear} \urcorner &= \mathit{linear}\, \ulcorner e \urcorner \\ \ulcorner \Lambda x \,.\, e \text{ linear in } x \urcorner &= \mathit{linx}(\lambda x{:}\mathit{exp}. \ulcorner e \urcorner) \end{aligned}$$

[1] We use object-logic capital Λ to avoid confusion with λ in the meta-logic.

The two judgments are realized as follows:

$$
\begin{aligned}
linlam : {} & linear(lam(\lambda x \,.\, E\ x)) \\
& \leftarrow linx(\lambda x \,.\, E\ x) \\
& \leftarrow (\forall y\!:\!exp.\, linear(y) \rightarrow linear(E\ y)). \\
linapp : {} & linear(app\ E_1\ E_2) \\
& \leftarrow linear(E_1) \\
& \leftarrow linear(E_2).
\end{aligned}
$$

$$
\begin{aligned}
linxx : {} & linx(\lambda x \,.\, x). \\
linxlm : {} & linx(\lambda x \,.\, lam(\lambda y \,.\, E\ x\ y)) \\
& \leftarrow (\forall y\!:\!exp.\, linx(\lambda x \,.\, E\ x\ y). \\
linxapp_1 : {} & linx(\lambda x \,.\, app\ (E_1\ x)\ E_2) \\
& \leftarrow linx(\lambda x \,.\, E_1\ x). \\
linxapp_2 : {} & linx(\lambda x \,.\, app\ E_1\ (E_2\ x)) \\
& \leftarrow linx(\lambda x \,.\, E_2\ x).
\end{aligned}
$$

This is clearly a decision procedure, which can be complemented; an expression is *not* linear if there is some functional sub-term which either does not use one of its argument or uses it more than once. The following is obtained via the clause complementation algorithm described in [1].

First the *normalization* phase: since the program is already *left-linear* and does not have any *dynamic input variable*, the first two steps are skipped. Then, the *full application* translation introduces the appropriate `vac` constraints in clauses `linxapp1,linxapp2`, while we $\top$-*normalize* `linxlm`:

$$
\begin{aligned}
linlam : {} & linear(lam(\lambda x \,.\, E\ x)) \\
& \leftarrow linx(\lambda x \,.\, E\ x) \\
& \leftarrow (\forall y\!:\!exp.\, linear(y) \rightarrow linear(E\ y)). \\
linapp : {} & linear(app\ E_1\ E_2) \\
& \leftarrow linear(E_1) \\
& \leftarrow linear(E_2).
\end{aligned}
$$

$$
\begin{aligned}
linxx : {} & linx(\lambda x \,.\, x). \\
linxlm' : {} & linx(\lambda x \,.\, lam(\lambda y \,.\, E\ x\ y)) \\
& \leftarrow (\forall y\!:\!exp.\, \top_{linx} \rightarrow linx(\lambda x \,.\, E\ x\ y)). \\
linxapp'_1 : {} & linx(\lambda x \,.\, app\ (E_1\ x)\ (E_2\ x)) \\
& \leftarrow vac(\lambda x \,.\, E_2\ x) \wedge linx(\lambda x \,.\, E_1\ x). \\
linxapp'_2 : {} & linx(\lambda x \,.\, app\ (E_1\ x)\ (E_2\ x)) \\
& \leftarrow vac(\lambda x \,.\, E_1\ x) \wedge linx(\lambda x \,.\, E_2\ x).
\end{aligned}
$$

$$
v : vac(\lambda x \,.\, E).
$$

We then proceed to *augmentation*, which inserts the necessary assumption complementation in clause `linxlm` and hereditarely in `linlam`, while everywhere else it is trivial:

$$
\begin{aligned}
aug_{\mathrm{D}}(linlam) : {} & linear(lam(\lambda x \,.\, E\ x)) \\
& \leftarrow linx(\lambda x \,.\, E\ x) \\
& \leftarrow (\forall y\!:\!exp.\, (\neg linx(\lambda x \,.\, y) \wedge linear(y)) \rightarrow linear(E\ y)). \\
linapp : {} & linear(app\ E_1\ E_2) \\
& \leftarrow linear(E_1)
\end{aligned}
$$

$$\leftarrow linear(E_2).$$

$$\begin{aligned}
linxx &: linx(\lambda x\,.\,x).\\
aug_{\mathrm{D}}(linxlm') &: linx(\lambda x\,.\,lam(\lambda y\,.\,E\ x\ y))\\
&\quad \leftarrow (\forall y\!:\!exp.\,\neg linx(\lambda x\,.\,y) \rightarrow linx(\lambda x\,.\,E\ x\ y)).\\
linxapp'_1 &: linx(\lambda x\,.\,app\ (E_1\ x)\ (E_2\ x))\\
&\quad \leftarrow vac(\lambda x\,.\,E_2\ x) \wedge linx(\lambda x\,.\,E_1\ x).\\
linxapp'_2 &: linx(\lambda x\,.\,app\ (E_1\ x)\ (E_2\ x))\\
&\quad \leftarrow vac(\lambda x\,.\,E_1\ x) \wedge linx(\lambda x\,.\,E_2\ x).
\end{aligned}$$

$$v : vac(\lambda x\,.\,E).$$

We can not apply the $\mathrm{Not_D}$ rules, to complement def(linear).

$$\mathrm{Not_D}(\mathrm{def}(linear)) = \mathrm{Not_D}(linlam) \vee \mathrm{Not_D}(linapp),\ \text{where:}$$

$$\begin{aligned}
&\mathrm{Not_D}(linlam) =\\
&\quad \neg linear(app\ E_1\ E_2) \wedge\\
&\quad \neg linear(lam(\lambda x\,.\,E\ x)) \leftarrow \neg linx(\lambda x\,.\,E\ x) \vee\\
&\qquad\qquad (\forall y\!:\!exp.\,(\neg linx(\lambda x\,.\,y) \wedge linear(y)) \rightarrow \neg linear(E\ y)).\\
&\mathrm{Not_D}(linapp) =\\
&\quad \neg linear(lam(\lambda x\,.\,E\ x)) \wedge\\
&\quad \neg linear(app\ E_1\ E_2) \leftarrow \neg linear(E_1) \vee \neg linear(E_2).
\end{aligned}$$

The complementation of def(linx) requires the use of the pre-compiled definition of `strict`$= \neg$`vac`:

$$\mathrm{Not_D}(\mathrm{def}(linx)) = \mathrm{Not_D}(linxx) \vee \mathrm{Not_D}(linxlm) \vee \mathrm{Not_D}(linxapp_1) \vee \mathrm{Not_D}(linxapp_2),\ \text{where:}$$

$$\begin{aligned}
&\mathrm{Not_D}(linxx) =\\
&\quad \neg linx(\lambda x\,.\,app\ (E_1\ x)\ (E_2\ x)) \wedge \neg linx(\lambda x\,.\,lam(\lambda y\,.\,(E\ x\ y))).\\
&\mathrm{Not_D}(linxlm) =\\
&\quad \neg linx(\lambda x\,.\,x) \wedge \neg linx(\lambda x\,.\,app\ (E_1\ x)\ (E_2\ x)) \wedge\\
&\quad \neg linx(\lambda x\,.\,lam(\lambda y\,.\,(E\ x\ y))) \leftarrow (\forall y\!:\!exp.\,\neg linx(\lambda x\,.\,y) \rightarrow \neg linx(\lambda x\,.\,E\ x\ y)).\\
&\mathrm{Not_D}(linxapp_1) =\\
&\quad \neg linx(\lambda x\,.\,x) \wedge \neg linx(\lambda x\,.\,lam(\lambda y\,.\,(E\ x\ y))) \wedge\\
&\quad \neg linx(\lambda x\,.\,app\ (E_1\ x)\ (E_2\ x)) \leftarrow strict(\lambda x\,.\,E_2\ x) \vee \neg linx(\lambda x\,.\,E_1\ x)).\\
&\mathrm{Not_D}(linxapp_2) =\\
&\quad \neg linx(\lambda x\,.\,x) \wedge \neg linx(\lambda x\,.\,lam(\lambda y\,.\,(E\ x\ y))) \wedge\\
&\quad \neg linx(\lambda x\,.\,app\ (E_1\ x)\ (E_2\ x)) \leftarrow strict(\lambda x\,.\,E_1\ x) \vee \neg linx(\lambda x\,.\,E_2\ x)).
\end{aligned}$$

and `strict` is as follows:

$$\begin{aligned}
sid &: strict(\lambda x\,.\,x).\\
sapp_1 &: strict(\lambda x\,.\,app\ (E_1\ x)\ (E_2\ x)) \leftarrow strict(\lambda x\,.\,E_1\ x).\\
sapp_2 &: strict(\lambda x\,.\,app\ (E_1\ x)\ (E_2\ x)) \leftarrow strict(\lambda x\,.\,E_2\ x).\\
slam &: strict(\lambda x\,.\,lam\ (\lambda y\,.\,E\ x\ y)) \leftarrow (\forall z\!:\!exp.\,strict(\lambda x\,.\,E\ x\ z)).
\end{aligned}$$

Note the crucial difference between the definition of `strict` and `linxx` lies in that in the latter the clauses for application are *not* fully. applied.

The final step is the elimination of disjunctions via unification. We exemplify this only for $\mathrm{Not}_{\mathrm{D}}(\mathit{linxapp}_1) \vee \mathrm{Not}_{\mathrm{D}}(\mathit{linxapp}_2)$. Since this operation is obviously commutative and associative, the process can be performed incrementally and in any order.

$$
\begin{aligned}
&\mathrm{Not}_{\mathrm{D}}(\mathit{linxapp}_1) \vee \mathrm{Not}_{\mathrm{D}}(\mathit{linxapp}_2) = \\
&\quad \neg \mathit{linx}(\lambda x \,.\, x). \\
&\quad \neg \mathit{linx}(\lambda x \,.\, \mathit{lam}(\lambda y \,.\, E\ x\ y)). \\
&\quad \neg \mathit{linx}(\lambda x \,.\, \mathit{app}\ (E_1\ x)\ (E_2\ x)) \\
&\qquad\quad \leftarrow \mathit{strict}(\lambda x \,.\, E_2\ x) \vee \neg \mathit{linx}(\lambda x \,.\, E_1\ x)) \\
&\qquad\quad \leftarrow \mathit{strict}(\lambda x \,.\, E_1\ x) \vee \neg \mathit{linx}(\lambda x \,.\, E_2\ x)).
\end{aligned}
$$

The final negative program is as follows, where we have renamed and brought clauses to LF-like syntax.

$$
\begin{aligned}
\mathit{non\_linlam}_1 :\ & \mathit{non\_linear}(\mathit{lam}(\lambda x \,.\, E\ x)) \\
& \leftarrow \mathit{non\_linx}(\lambda x \,.\, E\ x). \\
\mathit{non\_linlam}_2 :\ & \mathit{non\_linear}(\mathit{lam}(\lambda x \,.\, E\ x)) \\
& \leftarrow (\forall y{:}\mathit{exp}.\, (\mathit{non\_linx}(\lambda x \,.\, y) \wedge \mathit{linear}(y)) \rightarrow \mathit{non\_linear}(E\ y)). \\
\mathit{non\_linapp}_1 :\ & \mathit{non\_linear}(\mathit{app}\ E_1\ E_2) \\
& \leftarrow \mathit{non\_linear}(E_1). \\
\mathit{non\_linapp}_2 :\ & \mathit{non\_linear}(\mathit{app}\ E_1\ E_2) \\
& \leftarrow \mathit{non\_linear}(E_2).
\end{aligned}
$$

$$
\begin{aligned}
\mathit{non\_linxlm} :\ & \mathit{non\_linx}(\lambda x \,.\, \mathit{lam}(\lambda y \,.\, E\ x\ y)) \\
& \leftarrow (\forall y{:}\mathit{exp}.\, \mathit{non\_linx}(\lambda x \,.\, y) \rightarrow \mathit{non\_linx}(\lambda x \,.\, E\ x\ y)). \\
\mathit{non\_linxapp}_1 :\ & \mathit{non\_linx}(\lambda x \,.\, \mathit{app}\ (E_1\ x)\ (E_2\ x)) \\
& \leftarrow \mathit{strict}(\lambda x \,.\, E_1\ x) \\
& \leftarrow \mathit{strict}(\lambda x \,.\, E_2\ x). \\
\mathit{non\_linxapp}_2 :\ & \mathit{non\_linx}(\lambda x \,.\, \mathit{app}\ (E_1\ x)\ (E_2\ x)) \\
& \leftarrow \mathit{strict}(\lambda x \,.\, E_1\ x) \\
& \leftarrow \mathit{non\_linx}(\lambda x \,.\, E_1\ x). \\
\mathit{non\_linxapp}_3 :\ & \mathit{non\_linx}(\lambda x \,.\, \mathit{app}\ (E_1\ x)\ (E_2\ x)) \\
& \leftarrow \mathit{strict}(\lambda x \,.\, E_2\ x) \\
& \leftarrow \mathit{non\_linx}(\lambda x \,.\, E_2\ x). \\
\mathit{non\_linxapp}_4 :\ & \mathit{non\_linx}(\lambda x \,.\, \mathit{app}\ (E_1\ x)\ (E_2\ x)) \\
& \leftarrow \mathit{non\_linx}(\lambda x \,.\, E_1\ x) \\
& \leftarrow \mathit{non\_linx}(\lambda x \,.\, E_2\ x).
\end{aligned}
$$

We remark that the occurrence constraints can be *internalized* in a strict type theory as described in [3]. More details and proofs can be found in [2].

References

[1] A. Momigliano. Elimination of negation in a logical framework. In P. G. Clote and H. Schwichtenberg, editors, *Computer Science Logic*, volume 1862 of *Lecture Notes in Computer Science*, pages 410–415. Springer, 2000.

[2] A. Momigliano. *Elimination of Negation in a Logical Framework*. PhD thesis, Carnegie Mellon University, 2000.

[3] A. Momigliano and F. Pfenning. The relative complement problem for higher-order patterns. In D. D. Schreye, editor, *Proceedings of the 1999 International Conference on Logic Programming (ICLP'99)*, pages 389–395, La Cruces, New Mexico, 1999. MIT Press.