Proceedings of the Workshop on Linear Logic and Logic Programming, Washington, DC, 14 November1992 (following the 1992 Joint International Conference and Symposium on Logic Programming). Ed. by Dale Miller (University of Pennsylvania Department of Computer and Information Science Technical Report No. MS-CIS-92-80), https://repository.upenn.edu/cis_reports/301

Rules of definitional reflection in logic programming

Peter Schroeder-Heister
Universität Tübingen, Wilhelm-Schickard-Institut
Sand 13, 7400 Tübingen, Germany
e-mail: schroeder-heister@mailserv.zdv.uni-tuebingen.de

Given a set $\mathbf{D}$ of clauses of the form

$$F \Rightarrow A,$$

where F is a formula of some logic and A is an atom, it is natural to extend the sequent calculus for that logic by a rule like

$$\frac{\Gamma \vdash F}{\Gamma \vdash A} \; (\vdash \mathbf{D}),$$

yielding a logic over $\mathbf{D}$. This idea has been used in proof-theoretic interpretations and extensions of definite Horn clause programming, notably λ-Prolog, by giving a computational reading to $(\vdash \mathbf{D})$, which corresponds to resolution if the clauses in $\mathbf{D}$ are of a particular form.

In systems like GCLA, a principle dual to $(\vdash \mathbf{D})$ is considered in addition, yielding a fully symmetric sequent calculus. It is called "definitional reflection" since it is based on reading the database $\mathbf{D}$ as a definition. There are two main options for formulating definitional reflection. The rule on which GCLA is based is the following:

$$\frac{\{\Gamma, F\sigma \vdash G : F \Rightarrow B \in \mathbf{D} \textit{ and } A = B\sigma\}}{\Gamma, A \vdash G} \; (\mathbf{D} \vdash).$$

An alternative rule which has been considered by Eriksson and which seems also to be the one Girard is favoring, has the following form:

$$\frac{\{\Gamma\sigma, F\sigma \vdash G\sigma : F \Rightarrow B \in \mathbf{D} \textit{ and } \sigma = mgu(A, B)\}}{\Gamma, A \vdash G} \; (\mathbf{D} \vdash)^*.$$

As they stand, $(\mathbf{D} \vdash)^*$ is stronger than $(\mathbf{D} \vdash)$ (in the non-propositional case) - a standard example being the derivations of the axioms of ordinary first-order equality theory. Computationally, however, they rest on different intuitions. The first rule considers free variables as *existentially* quantified from outside, for which an appropriate substitution has to be computed. The second rule considers them as *universally* quantified from outside rather than something for which an substitution has still to be found. By means of unification it takes into account all possible substitution instances of the atom A, which can be inferred according to the given definition $\mathbf{D}$, thus corresponding to some kind of ω-rule.

Therefore, the extension of logic programming systems by computational variants of $(\mathbf{D} \vdash)$ and $(\mathbf{D} \vdash)^*$ leads to conceptually different approaches. A combination of $(\mathbf{D} \vdash)$ and $(\mathbf{D} \vdash)^*$ with both existential and universal variables, as proposed by Eriksson, would be a most desirable feature of a logic programming system with definitional reflection. There are certain algorithmic problems involved in such a combination that have still to be solved.

In any case, whether one considers $(\mathbf{D} \vdash)$ or $(\mathbf{D} \vdash)^*$ or a combination of both, cut-elimination fails for the full system but holds if the definition $\mathbf{D}$ does not contain implications in clause bodies or if the underlying logic is contraction-free (e.g., linear). We argue that the failure of cut-elimination is a matter of the definition $\mathbf{D}$ considered rather than a defect of the underlying logic.