Hypothetical Reasoning and Definitional Reflection in Logic Programming

Peter Schroeder-Heister *

Universität Tübingen/SNS
Biesingerstr. 10, 7400 Tübingen, Germany

This paper describes the logical and philosophical background of an extension of logic programming which uses a general schema for introducing assumptions and thus presents a new view of hypothetical reasoning. The detailed proof theory of this system is given in [7], matters of implementation and control of the corresponding programming language GCLA with detailed examples can be found in [1, 2]. In Section 1 we consider the local rule-based approach to a notion of atomic consequence as opposed to the global logical approach. Section 2 describes our system and characterises the inference schema of definitional reflection which is central for our approach. In Section 3 we motivate the computational interpretation of this system. Finally, Section 4 relates our approach to the idea of logical frameworks and the way elimination inferences for logical constants are treated therein, and thus to the notions of logic and structure. It shows that from a certain perspective, logical reasoning is nothing but a special case of reasoning in our system.

1 Local and global consequence

If one poses an atom A as a query with respect to a definite Horn clause program P, this is normally understood as asking whether there is a substitution θ such that $A\theta$ can be inferred from P, and for which substitutions this holds. Symbolically, we may represent this as

$$(?\theta)\ P \models A\theta, \tag{1}$$

where $\models$ denotes first-order logical consequence. Because first-order logic is complete, we may alternatively write

$$(?\theta)\ P \vdash_L A\theta, \tag{2}$$

where $\vdash_L$ denotes derivability in some formalization of first-order logic. Expression (1) would represent a model-theoretic interpretation, (2) a proof-theoretic

* I would like to thank Michael Morreau for helpful suggestions.

interpretation of definite Horn clause programming. In both cases, the program P is considered a collection of formulae of a certain form, viz.

$$(\forall)(A_1 \wedge \ldots \wedge A_n \supset A) \tag{3}$$

for atoms $A_1, \ldots, A_n, A$, or some equivalent of it (here $\forall$ denotes universal closure). We call this the *clauses-as-formulae* view of logic programs. Since program clauses are considered hypotheses or axioms with respect to which something is proved, we may also speak of the *clauses-as-axioms* view. If one wants to apply proof-theoretic methods to prove, e.g., the soundness and completeness of SLD-resolution, one might consider for L a system which is proof-theoretically easily tractable such as Gentzen's sequent calculus LK or some appropriate subsystem thereof. This is the way the theory of logic programming is presented in Beeson [3] and completeness is proved.

It is then easy to extend logic programming to cover hypothetical queries posing A with respect to a hypothesis H and asking for which substitution θ the atom $A\theta$ can be inferred from $H\theta$ with respect to P, symbolically

$$(?\theta)\ H\theta, P \vdash_L A\theta. \tag{4}$$

The hypothesis $H\theta$ is simply put on the left side of the turnstile in addition to the program P. Extensions of logic programing, which treat hypothetical reasoning in that way, have been developed by Gabbay and Reyle [5], and, in a sequent-style framework, by Miller [11] and Beeson [3].

We propose a different approach to hypothetical reasoning, considering a program P to be a set of rules rather than a set of axioms. As rules, which may be written as

$$A_1, \ldots, A_n \Rightarrow A \tag{5}$$

instead of (3), clauses define themselves a notion of consequence $\vdash_P$. According to this *clauses-as-rules* view, instead of (2) we now interpret a query posing A as asking

$$(?\theta) \vdash_P A\theta,$$

and instead of (4), a hypothetical query is now interpreted as asking

$$(?\theta)\ H\theta \vdash_P A\theta. \tag{6}$$

If we consider $\vdash_P$ to be a local notion of consequence and $\vdash_L$ to be a global one, we may say that according to (4) program clauses are axioms with respect to a global notion of consequence whereas according to (6) they are rules defining a local notion of consequence. Concerning hypotheses, we may say that in (4) they are formally treated in the same way as program clauses (namely as assumptions with respect to global deducibility), whereas in (6) they are treated differently from clauses: The hypotheses as *assumptions* with respect to global deducibility, and the program clauses as *defining* local deducibility.

Unter a certain interpretation of $\vdash_P$ (see below), these two approaches are intertranslatable:

$$\ldots, P \vdash_L \ldots \ (\textit{derivability from } P \textit{ in } L) \tag{7}$$

and

$$\ldots \vdash_P \ldots \; (\textit{derivability in } P) \tag{8}$$

are equivalent (if rules of the form (5) are appropriately translated into formulae of the form (3) and vice versa). From this point of view, the difference between the clauses-as-axioms view and the clauses-as-rules view is analogous to that between Hilbert-style and Gentzen-style formalizations of logic, now appearing at the level of atoms rather than logical constants. The (global) derivability *from* P in L corresponds to the derivability *from* logical axioms using only the global rule of modus ponens (in the propositional case), whereas the (local) derivability *in* P corresponds to the derivability *using* logical rules which are specific for every logical constant. This leads to an important shift in perspective on logic programming, corresponding to the conceptual shift from Hilbert- to Gentzen-style calculi. In particular, it allows one to prove standard results of the theory of logic programming (such as the completeness of SLD-resolution) in a straightforward way, since derivations in the sense of $\vdash_P$ are closely related to SLD-derivations (such a proof is given in [7]).

Proof-theoretically, this means that logic programming basically belongs to the theory of atomic systems. According to this view logic programming is, literally speaking, not *logic* programming but programming with *atomic* rules. These rules can be translated into logical language, but this translation is conceptually secondary.

2 Definitional reflection

However, we propose an even stronger reading of $\vdash_P$, according to which

$$\ldots, P \vdash_L \ldots$$

and

$$\ldots \vdash_P \ldots$$

are no longer equivalent. This reading is based on a definitional view of logic programs. We look upon the clauses of a program as *definitions* of their heads. In contradistinction to program clauses, assumptions appearing to the left of the turnstile are not considered as contributing anything to the meaning of atoms. Whereas as a definition the program fixes the "world" one is dealing with in a particular context, assumptions are just hypotheses about what is the case in this world without changing it.

This idea will be captured by defining a consequence relation $\vdash_P$ by means of a sequent calculus.[1] The definitional reading of programs is determined by a specific inference schema of definitional reflection. This schema allows one to assume an atom A by reference to the program rules defining (= permitting to infer) A. If one uses this schema with respect to A, one refers in a specific

[1] Here "consequence relation" is not understood in Tarski's sense but quite unspecifically as a relation between assumptions and assertions.

way to what the *program* (definition) says about A and not to anything else we *assume* about A. For example, suppose $B \Rightarrow A$ is the only program rule by means of which A can be inferred, and we have derived

$$B \rightarrow A, B \vdash C,$$

then by definitional reflection we may pass to

$$B \rightarrow A, A \vdash C.$$

If there is no such program sule, we cannot perform this step, although the assumption $B \rightarrow A$ seems to say the same about A as does the program rule $B \Rightarrow A$ (the precise inference schemata for definitional reflection and for $\rightarrow$ are given below). The basic difference is that as a program rule $B \Rightarrow A$ is considered as defining A whereas as an assumption $B \rightarrow A$ is not so considered, and the inference schema of definitional reflection only refers to the definitional aspects (the meaning) of A.

The (local) consequence relation $\vdash_P$ generated by the program P is formally defined as follows:

We use A, B, C for atoms, F and G for implicational formulae built up from atoms by means of implication $\rightarrow$ (including atoms as a limiting case), and X, Y, Z for finite sets of implicational formulae. All letters may have subcripts. A *sequent* has the form

$$X \vdash F,$$

a *clause* or *program rule* the form

$$X \Rightarrow A.$$

Expressions like $X, Y \vdash F$ or $X, F \Rightarrow A$ are understood in the obvious way. A program P is a finite set of clauses. We write $X \vdash_P F$ to express that the sequent $X \vdash F$ is derivable in the sequent calculus with respect to a fixed program P.

The sequent system has the following three program-independent inference schemata:

$$(I) \quad \frac{}{X, A \vdash A}$$

$$(\vdash \rightarrow) \quad \frac{X, F_1 \vdash F_2}{X \vdash F_1 \rightarrow F_2}$$

$$(\rightarrow \vdash) \quad \frac{X \vdash F_1 \quad X, F_2 \vdash F}{X, F_1 \rightarrow F_2 \vdash F}.$$

These inference schemata constitute a Gentzen sequent-style implicational calculus. One also could give schemata for other operators such as conjunction or universal quantification. For simplicity (as regards the computational interpretation of the system) we restrict ourselves to implication. These operators need not be read as logical constants in the narrower sense, since they can be used in bodies of rules to *define* logical constants. We would rather prefer to speak of "structural" operators or connectives as opposed to logical ones. So we call $\rightarrow$ a "structural implication" to be distinguished from "logical implication" $\supset$. The reason for introducing $\rightarrow$ at all is that implications in the bodies of rules strongly increase the expressive power of logic programming, especially in connection with definitional reflection. Without $\rightarrow$ one would lose the power that one has in logical rules which allow discharge of assumptions. (For the relationship between logic programming and logical rules and generally between rules and structure see §4 below.)

There are two schemata referring to the program P. The first one is the following:

$$(\vdash P) \quad \frac{(X \vdash F\sigma)_{F \in Y}}{X \vdash A\sigma}$$

for any clause $Y \Rightarrow A$ in P. It simply says that from a substitution instance of the premisses of a clause one may pass over to the corresponding substitution instance of its conclusion, i.e., it expresses closure under program rules. Since we look at programs as definitions, it may be called the schema of *definitional closure*. It is what one would normally expect as a schema for rule application. We use the label "$(\vdash P)$", since it operates on the right side of the turnstile.

The schema of *definitional reflection* $(P\vdash)$, which is characteristic of our approach to hypothetical reasoning, permits the introduction of an atom on the left side of the turnstile. It is a natural counterpart of the schema of closure under program rules and is defined as follows: Let for any atom A,

$$\mathbf{D}_P(A) := \{Y\sigma : \ A = B\sigma \ for \ a \ clause \ Y \Rightarrow B \ in \ P\}.$$

Here $\mathbf{D}_P(A)$ is to be read as "the definiens of A according to P". Then

$$(P\vdash) \quad \frac{(X, Z \vdash F)_{Z \in \mathbf{D}_P(A)}}{X, A \vdash F} \ provided \, \mathbf{D}_P(A\tau) = (\mathbf{D}_P(A))\tau \ for \ all \ substitutions \ \tau$$

This inference schema can informally be read as follows: If F follows from X and the definiens of A, then F follows from X and A itself. It may be motivated in the following way: Since P is considered a definition, the clauses $Y \Rightarrow B$ in P whose heads have A as a substitution instance (i.e., $A = B\sigma$ for some σ), define A. The $Y\sigma$ in $\mathbf{D}_P(A)$ then exhaust all possibilities of inferring A according to the program and thus represent the "meaning" of A. Therefore everything one obtains from every $Y\sigma$ is obtained from A itself.

The proviso for the application of $(P\vdash)$ is an invariance condition. What should not happen is that by further specializing A by means of substitution the definiens of A is enlarged. This guarantees that the inference schema $(P\vdash)$ is closed under substitution.

The schema $(P\vdash)$ is a natural counterpart of the schema $(\vdash P)$ and thus fits in a very natural way into the schemata of Right- and Left- introductions in sequent style systems. To give a deeper understanding of this duality it might be useful to formulate $(\vdash P)$ and $(P\vdash)$ in natural deduction style as introduction and elimination schemata for A. The schema $(\vdash P)$ then reads

$$(A - I) \quad \frac{Y\sigma}{A} \ Y \Rightarrow B \ \in P \ and \ A = B\sigma$$

and $(P\vdash)$ reads

$$(A-E)\quad \frac{A \quad \begin{pmatrix} Y\sigma \\ \vdots \\ F \end{pmatrix}_{(Y\Rightarrow B)\,\in\, P \text{ and } A=B\sigma}}{F},$$

where $\begin{pmatrix} Y\sigma \\ \vdots \\ F \end{pmatrix}_{\dots}$ means that there are derivations of F from $Y\sigma$ for every $Y\sigma$ fulfilling the given condition. It is obvious that this inference schema is modelled according to the schema of $\vee$-elimination in natural deduction. It is furthermore obvious that $(A-I)$ and $(A-E)$ represent introduction and elimination schemata for an atom A and not for the predicate p, if A is $p(t)$ for some term t. The minor premisses of $(A-E)$ may change completely if one changes from $p(t)$ to some $p(t')$, if t' is not a substitution instance of t. Thus $(A-E)$ is not specific for p but for the whole atom $p(t)$. This makes our schema differ fundamentally from Martin-Löfs elimination principle for predicates in his theory of iterated inductive definitions ([10]), where the minor premisses are only dependent on the predicate being eliminated and not on a particular instance thereof. In this sense our principle of definitional reflection is local and not a global induction principle as Martin-Löf's. It is more closely related to Lorenzen's inversion principle (see [9, 8]).[2] Obviously the natural deduction schema $(A-E)$ is not very useful from a computational point of view (i.e., for backward reasoning), since the major premiss A does not occur below the line.

To illustrate $(P\vdash)$ by an example, consider the following propositional program as an example: $P = \{p\Rightarrow s,\ q\Rightarrow s\}$. Then we have the following derivation:

$$\dfrac{\dfrac{\overline{p, q\to r\vdash p} \quad \overline{r, p, q\to r\vdash r}}{p, p\to r, q\to r\vdash r}(\to\vdash) \quad \dfrac{\overline{q, p\to r\vdash q} \quad \overline{r, q, p\to r\vdash r}}{q, p\to r, q\to r\vdash r}(\to\vdash)}{s, p\to r, q\to r\vdash r}(P\vdash),$$

which corresponds to $\vee$-elimination if s is $p\vee q$. (For the relationship of the schema of definitional reflection to elimination inferences in natural deduction see §4.)

Another example: Let $\bot$ be a 0-ary predicate which in P is given no definition, i.e., there is no clause with head $\bot$ in P (otherwise P is arbitrary). Then $\mathbf{D}_P(\bot) = \emptyset$, thus the set of premisses of $(P\vdash)$ is empty, so that we can trivially derive

$$\overline{X, \bot\vdash F}$$

for any X and F. This means that we have an intrinsic notion of falsity built into the system with the *ex falso quodlibet* as its characteristic feature.

[2] If the set $\mathbf{D}_P(A)$ is just a singleton $\{Y\sigma\}$, the schema $(P\vdash)$ actually allows to invert the clause $Y\Rightarrow B$ (with $B\sigma = A$) in the sense that $A\vdash Y\sigma$ becomes derivable.

If one takes away from the system the schema $(P\vdash)$ one obtains a system which is extensionally equivalent to the system QN-Prolog of Gabbay & Reyle [5], and to a certain subsystem of systems by Miller and by Beeson ([11, 3]). Conceptually, however, it is still different from them, since they are all based on the clauses-as-formulae view. It is the clauses-as-rules view and its different treatment of assumptions and programs, which makes definitional reflection possible and thus the full symmetry in the inference schemata of the sequent calculus.

3 Computational interpretation

We now give a computational interpretation of the sequent system we have described so far. This computational interpretation may be viewed as an operational semantics of a programming language. A description of such a language GCLA is given in [1, 2].

A *goal* is defined to be a finite set of sequents. Goals are denoted by capital Greek letters $\Delta, \Gamma, \Sigma, \Pi$. We say that Σ is *valid* with respect to the program P if for each sequent $X\vdash F$ in Σ, $X\vdash_P F$ holds. When proving a goal Σ as a query with respect to P, we ask for substitutions θ such that $\Sigma\theta$ is valid with respect to P. In the following, we describe in abstract terms a method of how to compute, given Σ, substitutions θ such that $\Sigma\theta$ is valid with respect to P. This method is partly based on a generalization of SLD-resolution. The abstract description is given by an inductive definition of the relation "σ is computable for Σ with respect to P", in short: $\langle\Sigma, \sigma, P\rangle$ or $\langle\Sigma, \sigma\rangle$ (since P is assumed to be fixed).[3] We give this inductive definition in terms of a formal system; i.e., if one has derived $\langle\Sigma, \sigma\rangle$ in this system, this is to mean that σ is computable for Σ with respect to P. We throughout use the fact that the inference schemata of the sequent calculus introduced in the previous section are closed under substitution.

In the following we state inference schemata and give in each case an intuitive motivation telling why the schema reflects a computation step with respect to the consequence relation $\vdash_P$. In each case, a step

$$\frac{\langle\Sigma_1, \sigma_1\rangle}{\langle\Sigma_2, \sigma_2\rangle}$$

corresponds to a computation step leading from a goal Σ_2 to a subsequent goal Σ_1, expressing that if σ_1 is computable for the goal Σ_1 then σ_2 is computable for the goal Σ_2. In steps where no bindings to variables are created during computation, σ_1 equals σ_2. If a substitution σ is computed at that step, then σ_2 is $\sigma\sigma_1$. This corresponds to the fact that during evaluation of a query, substitutions are created stepwise and are then composed.

Axioms are of the form

$$\langle\emptyset, \sigma\rangle$$

[3] The third component P may be important for a concrete programming language, where one allows one to change the program P in addition to adding or deleting assumptions. This is actually the case in GCLA.

for any substitution σ.
Motivation: If the goal is empty then for any substitution σ, nothing needs to be computed, so every substitution is correct.

The remaining inference schemata correspond to those given for the relation $\vdash_P$:

$$\frac{\langle \Sigma\sigma, \theta\rangle}{\langle \Sigma \mathbin{\dot\cup} \{X, A \vdash B\}, \sigma\theta\rangle} \; if \; A\sigma = B\sigma.$$

Motivation: Suppose the sequent $X, A \vdash B$ occurs as a subgoal of the considered goal $\Sigma \mathbin{\dot\cup} \{X, A \vdash B\}$, and A unifies with B. This means that by applying the substitution σ to the goal $\Sigma \mathbin{\dot\cup} \{X, A \vdash B\}$ one obtains $\Sigma\sigma \mathbin{\dot\cup} \{X\sigma, A\sigma \vdash A\sigma\}$. Since $X\sigma, A\sigma \vdash A\sigma$ can be obtained by (I), one may omit this sequent from the goal and continue with $\Sigma\sigma$ as the subsequent goal. Therefore, if some θ is computable for the subsequent goal $\Sigma\sigma$, $\sigma\theta$ is computable for the original goal $\Sigma \mathbin{\dot\cup} \{X, A \vdash B\}$, since σ is the additional substitution computed at this step.

$$\frac{\langle \Sigma \cup \{X, F_1 \vdash F_2\}, \theta\rangle}{\langle \Sigma \mathbin{\dot\cup} \{X \vdash F_1 \rightarrow F_2\}, \theta\rangle}$$

$$\frac{\langle \Sigma \cup \{X \vdash F_1\} \cup \{X, F_1 \vdash F\}, \theta\rangle}{\langle \Sigma \mathbin{\dot\cup} \{X, F_1 \rightarrow F_2 \vdash F\}, \theta\rangle}$$

Motivation: Obvious from $(\vdash\rightarrow)$ and $(\rightarrow\vdash)$. No substitution is computed at these steps.

$$\frac{\langle \Sigma\sigma \cup \{X\sigma \vdash F\sigma : F \in Y\}, \theta\rangle}{\langle \Sigma \mathbin{\dot\cup} \{X \vdash B\}, \sigma\theta\rangle} \; if \; Y \Rightarrow A \; is \; in \; P \; and \; A\sigma = B\sigma$$

Motivation: Suppose $X \vdash B$ occurs as a subgoal of the considered goal $\Sigma \mathbin{\dot\cup} \{X \vdash B\}$ and B unifies with the head A of a program clause $Y \Rightarrow A$. Then by applying the substitution σ to this goal one obtains $\Sigma\sigma \mathbin{\dot\cup} \{X\sigma \vdash A\sigma\}$. Since the sequent $X\sigma \vdash A\sigma$ can be obtained from $\{X\sigma \vdash F\sigma : F \in Y\}$ by $(\vdash P)$, one may replace it by $\{X\sigma \vdash F\sigma : F \in Y\}$ and continue with $\Sigma\sigma \cup \{X\sigma \vdash F\sigma : F \in Y\}$ as the subsequent goal. Therefore, if θ is computable for this subsequent goal, $\sigma\theta$ is computable for the original goal, since σ is the additional substitution computed at this step.

$$\frac{\langle \Sigma\sigma \cup \{Y, X\sigma \vdash F\sigma : Y \in \mathbf{D}_P(A\sigma)\}, \theta\rangle}{\langle \Sigma \mathbin{\dot\cup} \{A, X \vdash F\}, \sigma\theta\rangle} \; if \; \mathbf{D}_P(A\sigma\tau) = (\mathbf{D}_P(A\sigma))\tau \; for \; all \; \tau$$

If the proviso (i.e., $\mathbf{D}_P(A\sigma\tau) = (\mathbf{D}_P(A\sigma))\tau$ *for all* τ) is fulfilled for σ, we also say that σ is *A-sufficient.*
Motivation: Suppose $A, X \vdash F$ occurs as a subgoal of the considered goal $\Sigma \mathbin{\dot\cup} \{A, X \vdash F\}$ and σ is A-sufficient. Then by applying the substitution σ to this goal one obtains $\Sigma\sigma \mathbin{\dot\cup} \{A\sigma, X\sigma \vdash F\sigma\}$. Since σ is A-sufficient, the proviso for the application of $(P\vdash)$ with respect to $A\sigma$ is fulfilled, i.e., $A\sigma, X\sigma \vdash F\sigma$ can be obtained from $\{Y, X\sigma \vdash F\sigma : Y \in \mathbf{D}_P(A\sigma)\}$ by $(P\vdash)$. Thus we may replace it by $\{Y, X\sigma \vdash F\sigma : Y \in \mathbf{D}_P(A\sigma)\}$ and continue with $\Sigma\sigma \cup \{Y, X\sigma \vdash F\sigma : Y \in \mathbf{D}_P(A\sigma)\}$

as the subsequent goal. Therefore, if θ is computable for this subsequent goal, $\sigma\theta$ is computable for the original goal, since σ is the additional substitution computed at this step.

If one considers only definite Horn clause programs, i.e., programs with only atoms in bodies of clauses, and allows only goals of the form $\{\vdash A_1, \ldots, \vdash A_n\}$, then one only needs $(\vdash P)$ and the corresponding schema in the definition of computability (and axioms $\langle\emptyset, \sigma\rangle$, of course). This corresponds exactly to SLD-resolution, showing that computability in our sense extends SLD-resolution in a certain way. In our case bindings are created at two new places: In the evaluation of (I) and of $(P\vdash)$. In the case of (I) this is just unification of the succedent with one antecedent of a sequent. In the case of $(P\vdash)$ it means the computation of an A-sufficient substitution. A feasible algorithm for the computation of A-sufficient substitutions is decribed in [7]. It must be noted, however, that there is no unique minimal (that is, most general) A-sufficient substitution for every A.

It can be shown that computability is *sound* and *complete* in the following sense:

Completeness of computability: For any substitution σ, $\Sigma\sigma$ is valid with respect to P iff there is a τ which assigns the same terms to variables occurring in Σ as σ and which is computable for Σ with respect to P.

The proof (see [7]) proceeds by stepwise comparing the inductive definitions of $\vdash_P$ and of computability with respect ot P. Since the inference schemata in these inductive definitions can be completely separated, it also contains proofs of the completeness of standard SLD-resolution for definite Horn clause programs and of an extended notion of computability for the system without $(P\vdash)$. Of course, this is only the abstract, nondeterministic notion of completeness, as it is normally considered in the theory of logic programming.[4]

4 Definitional Reflection and Structural Frameworks

It was basic for the described approach to logic programming that programs as sets of rules are conceptually kept apart from inference schemata handling these rules. In this way certain inferences, particularly definitional reflection $(P\vdash)$, can specifically refer to these rules as a separate sort of objects. This makes our approach differ from proof-theoretic approaches where program clauses are treated as special initial sequents or as sets of assumption formulae.

The conceptual division between rules and inference schemata is fundamental for the idea of structural frameworks, too. A structural framework (see [15]) is characterized by a concept of "rule" and a set of inference schemata that describe which inferences can be performed given a database of rules. Following Gentzen's terminology, these inference schemata are called "structural", since they do not contain logical content. When dealing with logics, the logical content is given

[4] For soundness and completeness results of the notion of falsity in our systems with respect to finite failure see [7].

through the database of rules. However, the inference schemata of a structural framework go much beyond what Gentzen called "structural". They do not only regulate the way formulae in a sequent may be associated (such as Thinning or Contraction), but also the way rules are treated. Furthermore, they may contain schemata concerning a sort (or sorts) of implication which is then not considered logical but structural implication, i.e., some analogue to logical implication at the strutural level (corresponding to the comma, which is a structural analogue of logical conjunction at the structural level, i.e., a structural conjunction). They may also contain a structural generalization corresponding to universal quantification in logic. So a structural framework is a kind of "structural logic" which particularly describes the handling of a database of rules. Therefore the picture is the following:

Principles for structural conjunction and implication Principles for handling the database	Database of rules

whereas in Gentzen's approach we have

Principles for structural conjunction Principles for logical constants

.

In Gentzen the content which is now in the database is part of specific inference schemata dealing with logical constants.

Structural frameworks are particularly well-suited for the treatment of logics with restricted structural postulates. In permitting different families of conjunction-like connectives with different structural postulates assumed for them they are similar to Belnap's display logic (see [4]). However, the consideration of structural implication(s) in combination with the treatment of databases of rules which may contain such structural implications in their bodies extends this approach considerably. It allows us a uniform treatment of logical constants in varying structural environments (see [12]).

More important in the present context, however, is that this approach is entirely independent of whether the content of the database is logical or not. It works for specific logical rules as well as for rules dealing with atoms. Moreover, it can be made plausible that the inference schema $(P\vdash)$ of definitional reflection is a reasonable ingredient of a structural framework which is not specific for the reasoning with atoms, but treats logically compound formulae as well. In this sense the sequent calculus presented in §2 represents a structural framework.

To demonstrate this universal character of our system let us use it as a structural framework for intuitionistic propositional logic, taking as the database the following introduction rules for propositional operators:

$$p, q \Rightarrow p\wedge q \qquad p\Rightarrow p\vee q \qquad q\Rightarrow p\vee q \qquad p\rightarrow q\Rightarrow p\supset q.$$

Here p and q are variables for formulae built up from certain sentential letters by means of the operators $\wedge$, $\vee$, $\supset$ and $\bot$. (Remember that $\rightarrow$ is a structural implication to be distinguished from $\supset$.) These formulae are viewed as atoms in the sense of the sequent calculus of §2 - we just consider the propositional operators as functors transforming atoms into atoms, without requiring that atoms start with predicates. It is then obvious that by using $(\vdash P)$ the following inference schemata can be derived:

$$\frac{X\vdash p \quad X\vdash q}{X\vdash p\wedge q} \quad \frac{X\vdash p}{X\vdash p\vee q} \quad \frac{X\vdash q}{X\vdash p\vee q} \quad \frac{X\vdash p\rightarrow q}{X\vdash p\supset q},$$

where the last one is interadmissible with

$$\frac{X, p\vdash q}{X\vdash p\supset q}.$$

By using $(P\vdash)$ the following inference schemata are immediately obtained:

$$\frac{X, p, q\vdash F}{X, p\wedge q\vdash F} \quad \frac{X, p\vdash F \quad X, q\vdash F}{X, p\vee q\vdash F} \quad \frac{X, p\rightarrow q\vdash F}{X, p\supset q\vdash F},$$

where the last one is interadmissible with

$$\frac{X\vdash p \quad X, q\vdash F}{X, p\supset q\vdash F}.$$

Together with the fact that

$$\frac{X\vdash\bot}{X\vdash F}$$

is admissible (one has again to use $(P\vdash)$, if $X\vdash\bot$ is an axiom), this yields with $\neg p$ as $p\supset\bot$ an intuitionistic sequent calculus.

Its remarkable feature is that it was obtained by just taking a database of introduction rules, which by $(\vdash P)$ generated the right-introduction inferences and by $(P\vdash)$ the left-introduction inferences. A natural deduction version with elimination inferences instead of left-introduction inferences can also be obtained:

$$\frac{X\vdash p\wedge q \quad Y, p, q\vdash F}{Y, X\vdash F} \quad \frac{X\vdash p\vee q \quad Y, p\vdash F \quad Y, q\vdash F}{Y, X\vdash F} \quad \frac{X\vdash p\supset q \quad Y, p\rightarrow q\vdash F}{Y, X\vdash F}.$$

Here the first schema is equivalent to

$$\frac{X\vdash p\wedge q}{X\vdash p} \quad \frac{X\vdash p\wedge q}{X\vdash q}$$

and the third one to

$$\frac{X\vdash p\supset q \quad Y\vdash p}{Y, X\vdash q}.$$

This shows that the schema of definitional reflection is very closely related to the uniform pattern for elimination inferences for natural deduction proposed in [13]. However, whereas there a general metalinguistic schema was proposed to generate explicit rules like

$$p\vee q, p\rightarrow r, q\rightarrow r\Rightarrow r$$

$$p \supset q, (p \to q) \to r \Rightarrow r,$$

$(P\vdash)$ works at the object level, so that elimination inferences are intrinsically available. This seems to be a good explication of Gentzen's dictum that "the introductions represent, as it were, the 'definitions' of the symbols concerned, and the eliminations are no more, in the final analysis, than the consequences of these definitions" ([6]). Our definitional reading of logic programming may be viewed as a generalization of Gentzen's definitional view of logical introduction rules. It expresses the computational reading of logic according to which introduction rules are the computationally basic production rules whereas elimination inferences just make explicit what is contained in the introduction rules read as definitions.

Apart from that, consideration of systems with weaker structural postulates suggests that this is the only way to deal with elimination inferences. As shown in [12], explicit rules like those just mentioned for $\vee$ or $\supset$ do not work in that context. This has to do with the fact that assumptions discharged in elimination inferences may be embedded in an arbitrary structural context and cannot in general be moved out if certain structural postulates (such as Exchange or Thinning) are not available. An intrinsic schema like $(P\vdash)$ seems to be the only way of treating these logics in a structural framework and obtaining a uniform picture of them. This supports definitional reflection from a completely different point of view. Definitional reflection is a principle that is neither specific to logic nor to logic programming but applies to the whole area of a computational approach to inference and hypothetical reasoning - logical or extra-logical. This view extends to all structural postulates, not just to definitional reflection. Any framework with restricted structural postulates naturally gives a declarative semantics of a logic programming language. Properly understood, it is not just a framework for, e.g., contraction-free, linear or relevant *logic*, but for any database of rules (see [12]).

It should be mentioned that when generalizing structural frameworks to permit arbitrary databases of rules one loses Cut as a general principle.[5] Cut holds, if rules are restricted in such a way that they obey certain well-foundedness principles. This is the case for introduction rules for logical operators where only subformulae of the conclusion occur in the premisses. It is also the case if the database is a definite Horn clause program (i.e., without $\to$ in the premiss of a rule), but it may fail, if one goes beyond that. The reason is that, when (structural) implication is available, the premiss of a program rule may be of higher complexity than its conclusion, which destroys induction over the complexity of the Cut-formula. However, this should not to be seen as a disadvantage, but as reflecting the generality of our approach. That Cut holds is a property of programs that one is lucky to obtain in many cases, and not a restriction on permissible programs which has to be checked in advance.

[5] If one takes a natural deduction version of the structural framework, with $(A - E)$ instead of $(P\vdash)$ (see §2) and Modus Ponens instead of $(\to\vdash)$ (or a corresponding formulation in sequent-style natural deduction), one loses normalizability.

1. Aronsson, M., Eriksson, L.-H., Gäredal, A., Hallnäs, L. & Olin, P. The programming language GCLA: A definitional approach to logic programming. *New Generation Computing*, 7 (1990), 381-404.
2. Aronsson, M., Eriksson, L.-H., Hallnäs, L. & Kreuger, P. A survey of GCLA: A definitional approach to logic programming (this volume).
3. Beeson, M. Some applications of Gentzen's proof theory in automated deduction (this volume).
4. Belnap, N. D. Display logic. *Journal of Philosophical Logic*, 11 (1982), 375-417.
5. Gabbay, D.M. & Reyle, U. N-PROLOG: An extension of PROLOG with hypothetical implications: I., *Journal of Logic Programming*, 1 (1984), 319-355.
6. Gentzen, G. Untersuchungen über das logische Schließen. *Mathematische Zeitschrift*, 39 (1935), 176-210, 405-431, English translation in: M.E. Szabo (ed.), *The Collected Papers of Gerhard Gentzen*, Amsterdam: North Holland, 1969, 68-131.
7. Hallnäs, L. & Schroeder-Heister, P. A proof-theoretic approach to logic programming. *SICS Research Report*, no. 88005, 1988. To appear in revised form in *Journal of Logic and Computation.*
8. Hermes, H. Zum Inversionsprinzip der operativen Logik. In: A. Heyting (ed.), *Constructivity in Mathematics*, Amsterdam: North-Holland, 1961, 62-68.
9. Lorenzen, P. *Einführung in die operative Logik und Mathematik*, Berlin: Springer, 1955.
10. Martin-Löf, P. Hauptsatz for the intuitionistic theory of iterated inductive definitions. In: J. E. Fenstad (ed.), *Proceedings of the Second Scandinavian Logic Symposium*, Amsterdam: North Holland, 1971, 179-216.
11. Miller, D. A theory of modules for logic programming. In: *Proceedings of the 1986 Symposium on Logic Programming (Salt Lake City Utah)*, IEEE Computer Society Press, Washington, 1986.
12. Schroeder-Heister, P. The role of elimination inferences in a structural framework. In: G. Huet (ed.), *Proceedings of the Esprit BRA Logical Frameworks Workshop, Sophia Antipolis 1990.*
13. Schroeder-Heister, P. A natural extension of natural deduction. *Journal of Symbolic Logic*, 49 (1984), 1284-1300.
14. Schroeder-Heister, P. Logic programming with weak structural rules. In preparation.
15. Schroeder-Heister, P. *Structural Frameworks with Higher-Level Rules: Proof-Theoretic Investigations.* Habilitationsschrift. Universität Konstanz, 1987.