# Regular Search Spaces (I): Horn and Normal Clauses

by

Alberto Momigliano and Mario Ornaghi

December 1993

Report CMU-PHIL-45

Carnegie Mellon

Philosophy
Methodology
Logic

Pittsburgh, Pennsylvania 15213-3890

# Regular Search Spaces (I): Horn and Normal Clauses[1]

**Alberto Momigliano**
Department of Philosophy
Carnegie Mellon University
15213 Pittsburgh PA, USA
**mobile@lcl.emu.edu**

**Mario Ornaghi**
Dipartimento di Scienze delTInformazione
Universita' degli studi di Milano
Via Comelico 39/41, Milano, Italy
**ornaghi@imiucca.csi.unimi.it**

## 1 Introduction

This is the first part of an attempt to provide a proof-theoretic foundation for logic programming based on the notion of *regular search spaces,* a simple logical framework where many logics can be expressed and guaranteed to enjoy an analog of the very features that make Prolog successful. Very roughly, a system is *regular* if, during search, no backtracking on substitutions is required.

In this first report we limit ourselves to a proof-theoretic reconstruction of logic programming, both for definite and normal programs. This leads us to a better understanding of negation-as-failure *(NF)* [Cla78].

This approach permits anyway to single out some of the features, namely *regularity,* that we maintain relevant for any theoretically well-founded extension of Prolog, much in the spirit of [Mil91]. Eventually, in the second forthcoming part, we shall embark in a generalization and abstraction of the nice properties of 5ZrZ}-resolution leading to the full formulation of the theory of *regular search spaces.*

Historically, the great majority of the papers on logic programming, in particular on its extensions, has been carried out in a semantic way. Unfortunately these semantics, being most of the times limited to term models, tend to hide proof-theoretic contents. Moreover, the procedural aspects are traditionally expressed in a refutational style. We prefer a more direct approach that by-passes, where possible and meaningful, this pseudo semantic/refutational style. We feel that many features that are rather cluttered in this framework instead become natural consequences of a proof-theoretic reading.

Basically two proof-theoretic approaches have been pursued in the literature, as outlined in [Hal90] - see section 5 for a more detailed discussion:

[1]**A version of this document will appear in *Extensions of Logic Programming,* Dickhoff R. (ed.), Lecture Notes in Computer Science n.?, Springer-Verlag 1994.**

1. Clauses as axioms and some sequent calculus to infer goals [Mil91].
2. Clauses as rules [Hal90]: programs should be seen as set of inference rules (inductive definitions) for the derivation of (not necessarily ground) atoms.

We think of our approach as a blend of the two. We introduce the concept of ***most general proof tree*** (*mgpt*), which corresponds to a $SLD$-derivation; i.e. $\theta$ is a computed answer substitution for $P \cup \{\leftarrow G\}$ iff there exists a mgpt for $\theta G$ with axioms from $P$, with no open assumption. Mgpt's are $SLD$-derivations upside-down, where the root is the goal to be proven, the assumptions are the intermediate goals there in and $\theta$ is the restriction to the free variables of $G$ of the composition of mgu's along the branch. Mgpt's are based on the notion of ***axiom application rule*** ($AAR$), which easily generalizes to more complex definitions of clauses and goals. Then we begin to formulate the concept of ***regular search space*** for which search strategies like Prolog's are complete. Therewith, introducing $AAR$'s-systems for negative goals and rules, we offer an analysis of $NF$, which clarifies its intrinsic incompleteness due to the fact that, in general, it gives rise to a non-regular search space. Moreover, the $safeness$ condition on the selection function is lead back to the usual proviso on parameters of the $\exists$-left rule.

For regular programs $NF$ can provide correct bindings for open negative queries. Otherwise, regularity can be achieved, through *splitting*. This can be the basis of a simplified synthesis technique in a spirit close to [Bar90].

This report is organized as follows: in section (2) we develop the proof-theory of definite programs, based on the notion of ***most general proof tree*** and of ***continuation***. The following section (3) makes the connection between $AAR$'s and Prolog computations: soundness and completeness are proved w.r.t. success and finite failure. In section (4) we present our proof-theoretic reconstruction of $SLDNF$-resolution and we prove the latter to be sound w.r.t. the former. Regular splitting is proposed as a solution to the problem of evaluating open negative queries. Eventually (5) we conclude with a review of the other existing proof-theoretic studies and with a note on future work.

## 2 The $SLD$-System

We associate to a program a suitable set of axioms and we formulate a rule *sld* to apply such axioms, that we call the ***axiom-application rule*** ($AAR$) for the $SLD$-system.

**Definition 2.1** *sld* is the rule which applies Horn axioms $\forall(A_1 \wedge \cdots \wedge A_n \rightarrow B)$ to goals $\theta B$, giving rise to sequences of new goals $\theta A_1, \ldots, \theta A_n$. The result of an application will be represented by a proof configuration such as:

$$\frac{\theta A_1 \cdots \ \theta A_n \quad \forall(A_1 \wedge \cdots \wedge A_n \rightarrow B)}{\theta B}\, sld$$

where the ***major premise*** $\forall(A_1 \wedge \cdots \wedge A_n \rightarrow B)$ is the applied ***axiom***, the possibly empty sequence of the *minor premises* $9A\backslash,..., 6A_n$ represents the new goals and the *conclusion 9B* is the starting goal.

Note that if we have in mind a Horn program, facts give rise to configurations where the sequence of minor premises is empty i.e., $n = 0$.

Next we inductively introduce the notion of *proof tree* (pt), which corresponds to a branch of a *SLD-tree.*

**Definition 2.2** An atom A is a *proof tree.* If IIi :: 0Ai,..., $\text{II}_n$ :: $9A_n$ are proof trees, then the following is also a *proof tree:*

$$\frac{\begin{matrix}\mathbf{n}_x & \mathbf{n}_n \\ \theta A_1 \text{ --- } & 0A_n\end{matrix} \quad \forall(A_1 \wedge \cdots \wedge A_n \rightarrow B)}{6B}\, sld$$

where II :: A is the linear notation for a proof tree with root *A.* We say that a formula is an *assumption* of a proof tree if it is a minor premise in some leaf. The root of a proof tree is called its *consequence.* The *axioms* of a proof tree axe the ones appearing as major premises. A proof tree is a *proof* of *B* from a set *A* of axioms iff *B* is its consequence, its axioms belong to *A* and it has no assumption.

**Property 2.1** If II is a proof tree and *9* is a substitution, then 0(11) is a proof tree.

This property allows us to introduce the following *pre-ordering* (intuitively to be read as IIi is less general or more instantiated than II2) and *equivalence* relation among proof trees:

**Definition** 2.3 IIi $\leq$ $n_2$ iff there is a *6* such that $n_a = 0(\text{II}_2)$; IIi $\equiv$ $\text{II}_2$ iff $n_x \leq \text{II}_2$ and $n_2 \leq n_{i;}$ Cone(n) = {IT | IT $\leq$ II}.

Note that this ordering can be seen as a sort of lifting to proof trees of the usual subsumption ordering among substitutions; remark also that equivalent proof trees axe identical modulo renaming of variables.

Using $\leq$, we can give a notion of *most general proof tree* among similar trees, where similarity is characterized as follows.

Roughly speaking, two proof trees are *similar* if they can be derived, starting from the root, by two axiom-application sequences that apply the same axioms in the same order, but possibly involving different substitutions. Similarity is crucial, because axiom-application sequences represent derivations of an idealized interpreter, that searches for proofs *U :: 0C* starting from 'goals' *C.* We will show that *regularity,* namely the existence of a most general proof tree among similar trees, is the property which makes *SLD-Kke* search strategies complete .

Our analysis of the SI,jD-system is the starting point for a study of more general AAiJ-systems. Therefore we define similarity in the general case, where nothing is assumed on the number and on the behaviour of the AAR's of a system.

**Definition 2.4** An axiom/rule-occurrence in a pt II is a triple *(p,Ax,R)* such that *p* is a path from the root to a node where *Ax* is the major premise applied by $R$. We say that two proof trees $\Pi_1$, $\Pi_2$ are ***similar***, written $\Pi_1 \sim \Pi_2$, if either they are two atoms with the same predicate symbol or they have the same (non empty) set of axiom/rule-occurrences.

Since the $SLD$-system is endowed with just one rule, namely *sld*, we shall suppress reference to rule until Section (4)

Similarity is an equivalence relation. We will call ***similarity classes*** the corresponding equivalence classes. With $Sim(\Pi)$ we will indicate the similarity class induced by a proof tree $\Pi$. The following *regularity* property holds.

**Property 2.2** $\Pi_1 \sim \Pi_2$ iff there is a proof tree $\Pi$ such that $\Pi_1 \leq \Pi$ and $\Pi_2 \leq \Pi$.

**Proof.** The direction from right to left is trivial. Conversely, we proceed by induction on the number of axiom occurrences applied in $\Pi_1$, $\Pi_2$.
**Basis.** There are 0 axiom occurrences. In this case, $\Pi_1$ and $\Pi_2$ are two atoms with the same predicate symbol, say $r$. Then $\Pi_1 = r(\underline{t})$ and $\Pi_2 = r(\underline{t}')$; then $\Pi = r(\underline{x})$.
**Step.** There are $n+1$ axiom occurrences, $\{o_1, \ldots, o_{n+1}\}$. Let $o_k$ be a top axiom-occurrence of some axiom $Ax = \forall(A_1 \wedge \cdots \wedge A_n \rightarrow B)$, as shown in the following figure:

$$\Pi_1 = \begin{array}{c} \theta_1 A_1 \cdots \ \theta_1 A_n \quad Ax \\ \hline \theta_1 B \\ \Gamma_1 \end{array} \qquad\qquad \Pi_2 = \begin{array}{c} \theta_2 A_1 \cdots \ \theta_2 A_n \quad Ax \\ \hline \theta_2 B \\ \Gamma_2 \end{array}$$

$\Gamma_1$ is similar to $\Gamma_2$, since both have axiom occurrences $\{o_1, \ldots, o_{n+1}\} - \{o_k\}$; by inductive hypothesis, there is a proof tree $\Gamma$ such that $\Gamma_1 = \sigma_1\Gamma$ and $\Gamma_2 = \sigma_2\Gamma$. Let $H$ be the top formula occurring in $\Gamma$, in the place of $\theta_1 B$ in $\Gamma_1$ (and of $\theta_2 B$ in $\Gamma_2$); we have: $\theta_1 B = \sigma_1 H$ and $\theta_2 B = \sigma_2 H$. We can assume that $H$ and $B$ have disjoint sets of variables, so that: $(\theta_1 \cup \sigma_1)B = (\theta_1 \cup \sigma_1)H$ and $(\theta_2 \cup \sigma_2)B = (\theta_2 \cup \sigma_2)H$, hence there is a mgu $\delta$ of $B$ and $H$, and we have: $\theta_1 \cup \sigma_1 = \mu_1\delta$ and $\theta_2 \cup \sigma_2 = \mu_2\delta$, for some $\mu_1, \mu_2$. Then we can build the proof tree $\Pi$:

$$\begin{array}{c} \delta A_1 \cdots \ \delta A_n \quad Ax \\ \hline \delta H \\ \delta\Gamma \end{array}$$

$\Pi$ is similar to $\Pi_1$ and $\Pi_2$, and one easily sees that $\Pi_1 = \mu_1\Pi$ and $\Pi_2 = \mu_2\Pi$ (we can assume that the variables of $\Gamma$ do not occurr in $A_1, \ldots, A_n, B$). This concludes the proof of the induction step. □

Note that the above proof relies not only on the similarity of the two proof trees, but also on the fact that the application (by *sld*) of $Ax$ preserves comparability. This does not hold w.r.t. a more general kind of $AAR$'s, e.g. the ones related to negation as failure, as shown in section 4.

**Property 2.3** For every set $\mathcal{S}$ of similar proof trees, there is a $\Pi$ such that $\mathcal{S} \subseteq Cone(\Pi)$.

**Proof.** Assume, by absurdum, that for every $\Pi$ there is a $\Pi^*$ in $\mathcal{S}$ such that $\Pi^* \not\leq \Pi$ and let $\Pi_0$ be a pt in the similarity class containing $\mathcal{S}$: there is a pt

Ill in *S* such that III % $n_0$. By regularity they have a similar upper bound, say Ai. Note that Ai cannot be a renaming of IIo. Consider II2 ^ Ai: by the same reasoning, they axe bounded by A2. Iterating the process we create an infinite chain $n_o$,Ai$A_2$..., where $n_0$ = #oAi, $A_x$ = #i$A_2$,... But this is not possible, since the term complexity order on pt's is well-founded. Q

**Property 2.4** For every II, Cone(II) $\subseteq$ Sim(II).

The similarity relation allows us to introduce the notion of *most general proof tree,* which corresponds to a SLD-branch where only mgu's axe used.

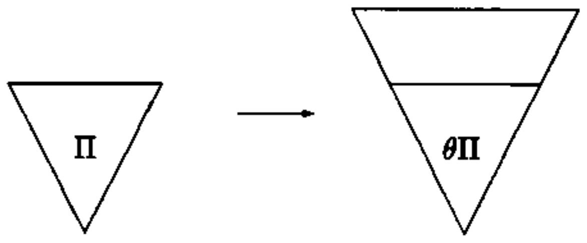
**Definition 2.5** We say that II is a *most general proof tree* (mgpt) if it is a *maximal* element among similar trees, i.e. Cone(II) = *Sim(JT).*

By the regularity property (2.2), every similarity class *Sim(IL)* contains a maximal element unique up to renaming. On the other hand minimal elements among similar trees axe ground proof trees, but in general there axe incomparable minimal elements with respect to $\leq$.

Now we target the formalization of how a Prolog computation can be extended, that is we try to capture the idea of the resolvents of a given goal.

**Definition 2.6** II2 is a *continuation* of III, denoted Eli ^ II2, iff there is an initial subtree II3 of $n_2$, that is with the same root, s.t. II3 $\leq$ III.

Assuming that the root is the goal to be proved, a continuation is a possible (not necessarily single-step) extension of a 5L2)-derivation, i.e. in our language, of the (open) assumptions of the goal to be proven. This extension may introduce new substitutions, as shown in the following picture, where a proof tree II is continued into a bigger proof tree containing 011 as an initial subtree.



The :$\leq$ relation is clearly a partial ordering w.r.t. the equivalence relation $\equiv$ and the immediate successors under this ordering of a pt essentially correspond to all possible resolvents of the current goal. Thus the notion of continuation captures the search aspect of logic programming. This characterization is further refined by the notion of most general continuation and canonical continuation and by their properties that we list below.

**Property 2.5** II $\leq$ II' entails II ^ II'.

**Definition 2.7** Let II, II* be a pt's and II ^ II*. We say that II* is a *most general continuation (mgc)* of II iff, for every continuation II' of II similar to II*, II' $\leq$ II*.

**Property 2.6** For every continuation II' of II, there is a mgc II* of II such that II' $\leq$ IT.

**Definition 2.8** A one-step continuation *selecting* an assumption *Hi* and *applying* an axiom *Ax* of the form V(Ai A • • • A A* —• *B)* s.t. *aB* = *aHi* is of the following form:

$$\cdots \quad \cfrac{\boldsymbol{GA\backslash\ -\bullet\bullet\ erAk\ ,\ Ax}}{\boldsymbol{aHi}} \quad \cdots$$
$$\boldsymbol{aU}$$

We say that a one-step continuation is *canonical* iff *a* is a most general unifier (assuming the usual standardization apart).

The following relations hold between subsumption and continuation, embodying the reason why search may be conducted exclusively on mgc's or on mgpt's and no backtracking on substitutions is required, contrary to what we will see in the section (4).

**Property 2.7** If **Hi** $\leq$ $U_2$ and **IIi** $\underline{X}$ **n$_3$,** then **II$_2$** *d* n$_3$.

**Property 2.8** Let IT be a one step continuation of II. II' is a mgc of II iff it is a canonical continuation.

**Property 2.9** If II is a mgpt, then its mgc's are mgpt's. In particular, its canonical continuations are mgpt's.

We conclude this section considering the consequences of the above properties with respect to the following search problem. Let *A* be a set of axioms, *T(A)* the set of the pt's with axioms from ,4, and II :: *C* be a pt of *T{A).* Search for a continuation II* :: *QC* 6 *T(A)* of II, such that 11* is a proof. One easily sees that the notions of mgpt, mgc and all the related properties hold even though we consider *T(A),* instead of the class of all proof trees. Thus one obtains:

**Property 2.10** If IIi is a mgpt of *T(A),* then, for every II$_2$ such that n$_x$ ~ II$_2$, if there is a II s.t. n$_2$ *d* II, then IIi ^ II.

**Proof.** Since II is a mgpt, then II$_2$ $\leq$ Hi. Apply (2.7). D

**Property 2.11** Let **II** be a mgpt of *T(A)* and *Hi* be an assumption of II. If for every canonical continuation II' of II selecting *Hi* there is no proof in *T(A)* continuing II', then there is no proof in *T(A)* continuing **II.**

**Proof.** By absurdum, let us suppose that there is a pt II* continuation of II. Then there is a subpt $\hat{\mathbf{n}}$ of II* that is a one step continuation of II selecting *H{.* Hence ft is similar to n'. But, by (2.9), n' is a mgpt and, by (2.10), n* is a continuation of n'. D

Property (2.10) more properly refines what we mean by "avoiding backtracking on substitutions". (2.11) shows that, by using canonical (i.e. most general) continuations, during the search the selection of the assumption *Hi* may be completely non deterministic. (2.10) and (2.11) allow also to state that a search strategy like the one of Prolog is complete in the search spaces for $T(\mathcal{A})$. But (2.10) and (2.11) are based on the ***regularity property*** (2.2), which may not hold in an arbitrary search space. This remark will be the basis of a next document, where we discuss *regular search spaces* (i.e. search spaces satisfying (2.2)) for a very general kind of axioms. In next section we discuss the case of Horn axioms.

## 3 Prolog Computations and the $SLD$-System

Eventually we link the $SLD$-system to Horn programs: to a program $P$ corresponds a set of axioms, indicated by $Ax(P)$, in the obvious way. For example, let us consider the following program $SUM$:

$$\begin{array}{l} sum(X,0,X). \\ sum(X,s(Y),s(Z)) :- sum(X,Y,Z) \end{array}$$

and the corresponding axioms $Ax(SUM)$:

$$\begin{array}{ll} Ax1 & \forall X.sum(X,0,X) \\ Ax2 & \forall X,Y,Z.sum(X,Y,Z) \rightarrow sum(X,s(Y),s(Z)) \end{array}$$

Examples of proof trees $\Pi_0$, $\Pi_1$ and $\Pi_2$ using $Ax(SUM)$ are:

$$\frac{\dfrac{sum(X,0,s(0))\ \ Ax2}{sum(X,s(0),s(s(0)))}\ \ Ax2}{sum(X,s(s(0)),s(s(s(0))))} \qquad \frac{\dfrac{\dfrac{Ax1}{sum(0,0,0)}\ \ Ax2}{sum(0,s(0),s(0))}\ \ Ax2}{sum(0,s(s(0)),s(s(0)))} \qquad \frac{\dfrac{\dfrac{Ax1}{sum(s(0),0,s(0))}\ \ Ax2}{sum(s(0),s(0),s(s(0)))}\ \ Ax2}{sum(s(0),s(s(0)),s(s(s(0))))}$$

$\Pi_0$ has assumption $sum(X,0,s(0))$ and consequence $sum(X,s(s(0)),s(s(s(0))))$; $\Pi_1$ and $\Pi_2$ are proofs (of their consequences). $\Pi_1$ and $\Pi_2$ are similar, since they have the same set of axiom occurrences $\{\langle[1],Ax2\rangle,\langle[0,1],Ax2\rangle,\langle[0,0,1],Ax1\rangle\}$ (paths are represented by sequences of numbers, in the usual way); the following proof tree is the mgpt in their similarity class:

$$\frac{\dfrac{\dfrac{Ax1}{sum(X,0,X)}\ \ Ax2}{sum(X,s(0),s(X))}\ \ Ax2}{sum(X,s(s(0)),s(s(X)))}$$

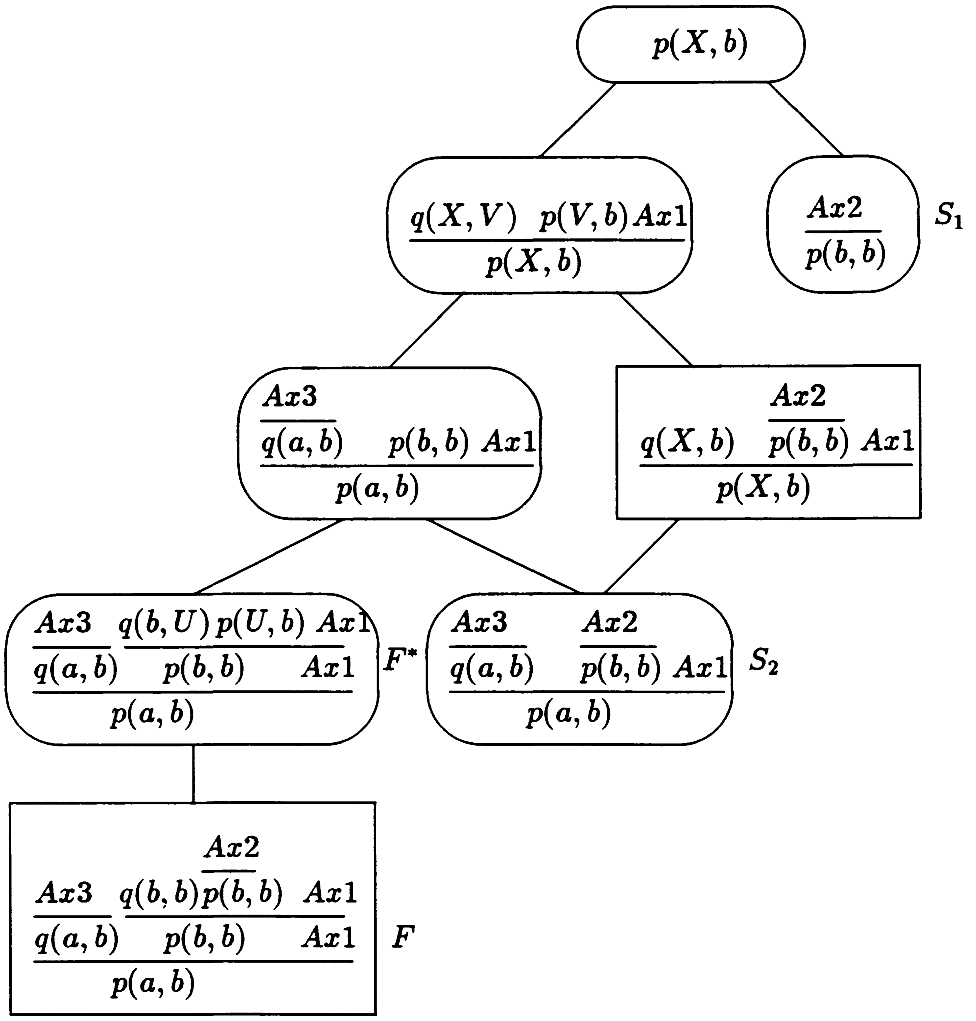
Now we address the following problem: given a program $P$ and an atomic formula $C$, search for a proof $\Pi :: \theta C$. From now on, in this section, we will call $C$ a *goal*. We give the following definition.

**Definition 3.1** Consider a program $P$ and a goal $C$ and let $T(P)$ be the set of the proof trees with axioms from $Ax(P)$ and $T(P,C)$ be the set of the proof trees $\Pi :: \theta C$ with axioms from $Ax(P)$. Define $Gen(P)$ and $Gen(P,C)$ to be the corresponding sets of mgpt's.

**Example 1** Let us consider the following program $P^2$ (see [Llo87], pp. 56-7).

$$\begin{array}{ll} Ax1 & p(X,Z) \leftarrow q(X,Y), p(Y,Z). \\ Ax2 & p(X,X). \\ Ax3 & q(a,b). \end{array}$$

Then these are all finite trees in $Gen(P,C)$ for the goal $p(X,b)$:



$p(X,b)$
$q(X,V)\ \ p(V,b)\ Ax1$ / $p(X,b)$
$Ax2$ / $p(b,b)$
$S_1$
$Ax3$ / $q(a,b)\ \ p(b,b)\ Ax1$ / $p(a,b)$
$Ax2$ / $q(X,b)\ \ p(b,b)\ Ax1$ / $p(X,b)$
$Ax3\ \ q(b,U)\ p(U,b)\ Ax1$ / $q(a,b)\ \ p(b,b)\ \ Ax1$ / $p(a,b)$
$F^*$
$Ax3\ \ Ax2$ / $q(a,b)\ \ p(b,b)\ Ax1$ / $p(a,b)$
$S_2$
$Ax2$ / $Ax3\ \ q(b,b)\ p(b,b)\ Ax1$ / $q(a,b)\ \ p(b,b)\ \ Ax1$ / $p(a,b)$
$F$

The success nodes $S_1$, $S_2$ contains *proofs* and the proof tree in the failure node $F$ has open assumptions, but no continuation. The oval boxes show the canonical continuations obtained by choosing the leftmost assumption. The rectangles show the canonical continuations selecting other assumptions. By (2.11), selecting other assumptions we obtain more proof trees, but the success set remains the same. Then we can stop at $F^*$, taking it as a failure node. ◇

As last step, consider the quotient $Gen(P,C)/\equiv$ under equivalence, that is renaming of variables. This passage is required since each resolution step considers

[2]From now on, we shall only quote the program, with the obvious axiomatic version.

variants of the input clauses to ease unification. We claim that *(Gen(P, C)/* $\equiv$, -$\leq$*)* corresponds to every *SLD-tvee* for PU {«— *C}* under each selection function, i.e., it contains every possible derivation. In fact, given opportune duplications of nodes with more than one parent, it can be looked at as a tree s. t.

- the root is *C*
- every axiom comes from *Ax(P)*
- every node is an equivalence class [II :: 0G], with II :: *OC* £ *Gen(P, C)*
- every child of a node is obtained by a canonical continuation applying an axiom of *Ax(P).*

**Proposition** 3.1 *0* is a computed answer substitution for *P* U {*— *C}* iff there is a II :: *OC* G *Gen(P, C)/*$\equiv$, with no open assumption.

**Proof.** Two straightforward inductions, respectively one on the length of the SLD-refutation and the other of the most general proof tree. •

As a corollary of the above construction, we obtain Lloyd's lemma on the independence of the computation rule (selection function), see [Llo87] theorem 9.2. This result establishes a sort of Church-Rosser property for SLD-resolution; if *P* U *{<— C}* is unsatisfiable then, whichever be the atoms selected in the inference steps, a refutation is reached. In this context it just stems from the fact that *Gen(P,C)* is a regular search space and from property (2.11) of regular search spaces (indeed, a SLjD-derivation of *P* U {<— *C}* with current goal <— *H\,... $H_n$* corresponds to a derivation of a proof tree II :: *OC* with assumptions i?i,... $\text{if}_n$; apply (2.11) to II :: *OC).* In this way, we can consider search trees built using a given selection function, where a *selection function F* is a function which associates to every proof tree II a fixed assumption in a leaf, called the *assumption selected by F* and denoted by F(II). We require that F(II) depends only on the axioms applied in II, i.e. that for any similar II, II', *F(IV)* and *F($U^f$)* select the same leaf.

**Definition** 3.2 Let *F* be a selection function. A *F-proof-sequence* is a (possibly infinite) sequence [$n_0$], [Hi],..., [$II_n$],... such that (for *i* $\geq$ 0) $II_{1+}$i is a canonical continuation of II, selecting the assumption *F(TLi).* An *F-search tree* for a program *P* and a goal *C* is a subtree of *(Gen(P, C)/* $\equiv$, -$\leq$*)* built from F-proof sequences in ***T(P,C).***

The independence result says that the proofs (i.e., pt's without assumptions) contained in *(Gen(P, C)/* $\equiv$, ^) and the ones contained in a *complete* F-search tree (namely a F-search tree such that the leaves contained therein have no continuations) are the same. Of course, the former contains more proof trees than the ones in *F-* search trees. In particular, it contains, for every rule F, the corresponding complete F-search tree.

This correspondence can be also applied w.r.t. finite failure: a *finitely failed* F-search tree for a program *P* and a goal *C* is just a finite and complete F-search tree for P, *C* such that all the leaves contained therein axe proof trees with open assumptions. Nevertheless, given the asymmetry of finite failure w.r.t. derivability, the independence result does not hold anymore: a $SLD$-tree can be finite under a selection function $F_1$ and infinite under another function $F_2$. We have to impose a *fairness* condition [Llo87] on the selection function to ensure that we find a finitely failed tree if one exists. Fairness can be characterized as follows.

**Definition 3.3** A F-proof-sequence is ***fair*** if it is finite or every (instance of an) atom appearing in it is selected by $F$. A $F$-search tree is ***fair*** if every F-proof-sequence is fair.

**Proposition 3.2** If there is a finitely-failed fair $F$-searchtree for a program $P$ and a goal $C$, then every $[\Pi :: \theta C] \in \langle Gen(P,C)/\equiv, \preceq\rangle$ has open assumptions.

**Proposition 3.3** There is a finitely-failed fair $F$-search tree for a program $P$ and a goal $C$ iff $P \cup \{\leftarrow C\}$ has a finitely-failed $SLD$-tree.

The notion of fairness can be made more explicit with the following example of fair selection function $F^*$: $F^*(\Pi)$ is the leftmost among the assumptions of $\Pi$ with minimal height.

## 4 The $SLDNF$-System

For our treatment of $SLDNF$-resolution, we need a new $AAR$ to apply the negative (only-if) part of the Completion axioms [Cla78], together with a suitable notion of negative goals. To distinguish positive and negative goals, we introduce a new symbol $\vdash_P$, used in a way similar to sequent calculi where $P$ is a list of existential parameters (eigenvariables), as originally suggested by Kleene. When the rules do not update this list, we shall simply omit it. Moreover, we need an $AAR$ to switch a negated positive goal into a negative goal and a negated negative goal into a positive one.

**Positive Goals and the Positive Rule.** Positive goals are of the form $\vdash L$, where $L$ is a literal. When $L$ is an atom $A$, the (+)-rule (or positive rule) allows to apply axioms corresponding to normal clauses, in the following way:

$$\frac{\vdash \theta L_1 \ \ldots \ \vdash \theta L_n, \ \ \forall(L_1 \wedge \cdots \wedge L_n \rightarrow A)}{\vdash \theta A}(+) \qquad \frac{\forall(A)}{\vdash \theta A}\ (+)$$

**Negative Goals and the Switch and Weakening Rules.** The negative goals are of the form $\Gamma \vdash$, where $\Gamma$ is a list of literals $L_1, \ldots, L_n$. There are several possible configurations: let us start from the switch ones. To switch a negated positive goal into a negative goal or a negated negative goal into a positive one we apply the switch rule (s):

$$\frac{\theta A \vdash, \ \ \forall(\neg A \rightarrow \neg B)}{\vdash \theta \neg B}\,(\mathrm{s}) \qquad \frac{\vdash \theta A\,, \ \ \forall(\neg B \rightarrow \neg A)}{\theta \neg B, \Gamma \vdash}\ (\mathrm{s})$$

Whenever $B$ = A, the rule is said to be *restricted* (to axioms of this form). Intuitively, the restricted switch rule can be seen as mirroring the evaluation steps of a goal selecting a negative literal under *NF:* the goal succeeds since h *A* fails (on the left hand side) and the goal fails since h *A* succeeds (on the right hand side). Note, however, that under the same proviso the rule corresponds to -»-R and -«-L in the sequent calculus with primitive negation. The weakening rule (w) is:

$$\frac{\text{T h,}\quad \text{V(-»A -+ } true)}{\neg A, \Gamma \vdash}\ (w)$$

To achieve a uniform treatment, we have expressed the rules (s) and (w) as AAiTs, even if they have a logical character.

**Negative Goals and the Negative Rule.** Next, we need a suitable negative rule (-) to apply the only-if part of the completion. First we show how (-) can be derived in minimal logic, starting from the identity and freeness axioms (see the Equality Theory in [Cla78]) together with the completed definitions of the predicates involved in a logic program (ibidem). Secondly, we explain (-) in a simplified case.

- *Rule derivation.* We derive our *AAR* (-) inside the minimal sequent calculus. Let us start with the *identity axioms* (idl),(id2) and the *substitution rules,* (ul) and (u2) which derive from the freeness axioms. Note that this equational theory can be shown to be complete and decidable ([Kun87], Theorem 5.5).

(idl) \~*t* = *t*

**(id2)** ***$t_l$=$t_2$,L($t_1$)\-L($t_2$)***

(ul) *t*\ = $t_2$ h *Eq((j)* if *a* is an idempotent mgu of *i, $t_2$ and *Eq(cr)* is the conjunction of the equations obtained from the bindings in *a*

(u2) *t*\ = $t_2$ h if no unifier of *t*\, $t_2$ exists

Now, assume the completed definition of a predicate *p(x)* of the form

**Va:(p(:r) <-• *3ui(x = $t_x$* A Mi) V *3$u_2${x =$t_2$* A *$M_2$))***

Here we are assuming that p(x) is defined by two clauses with body *M\,$M_2$* it is clear how to extend it to the general case. Consider the direction —>, that we call the *failure axiom* of *p(x).* Thanks to that, we can build proof trees of the form shown in the following, where a is a list of terms and, for the sake of this example, we suppose that <*j*\ is a mgu of a, *t*\ and that a, $t_2$ do not unify.

$$\frac{\dfrac{\dfrac{\dfrac{}{\;1\;}\ (\bullet)}{\underline{a} = \underline{t}_1, M_1, \Gamma \vdash_{\underline{u}_1}}\ (cut)}{\text{3*h(a = li A Mi), T h}}\ (\exists L) \qquad \dfrac{\dfrac{}{\underline{a} = \underline{t}_2, M_2, \Gamma \vdash_{\underline{u}_2}}}{\text{au\^{}a = } t_2 \text{ A } M_2\text{), } T \text{ h,}}\ (\exists L) \qquad FAx(p)}{p(\underline{a}), \Gamma \vdash}\ (f)$$

Let *FAx(p)* denote the failure axiom of *p(x):* (f) can be derived from p(a) h *3u_x(a_ = t_x* A Mi) V 3u₂(a = *t₂* A M2). The (3X) rule are iterated 3-left applications, where the involved eigenvariables are mentioned below the turnstile; (*) can be obtained by one or more applications of the identity rules, without introducing substitutions (in particular no substitution arises on the eigenvariables).

The above derivation lives in the minimal sequent calculus, thanks to the identity and failure axioms. Remark that we view the identity rules as a "logical" part that is common to every program to be executed in any calculus with unification: thus this proof depends in an essential way from the failure axiom *Fax(p).* It is summoned in the second of the following derived *AAR's.*

- *The Negative Rule.* (-) is a derived rules whose form *depends* on the following exhaustive unification possibilities, given the proviso on the eigenvariables decorating h:
  $\underline{a}_x$ unifies with $\underline{t}_x$ and $\underline{t}_2$;
  $\underline{a}_2$ unifies with $\underline{t}_x$ but not with $\underline{t}_2$;
  03 unifies with $\underline{t}_2$ but not with $\underline{t}_x$;
  04 does not unify with $\underline{t}_x$ nor with $\underline{t}_2$ .

$$\frac{\sigma_1 M_1, \sigma_1\Gamma \vdash_{\underline{u}_1,\underline{u}_2,P} \quad \sigma_2 M_2, \sigma_2\Gamma \vdash_{\underline{u}_1,\underline{u}_2,P} \quad FAx(p)}{p(\underline{a}_1), \Gamma \vdash_P}(-) \qquad \frac{<TiMi,(TiT\ \underline{K}_{lf}p \quad FAx(p)}{p(\underline{a}_2), \Gamma \vdash_P}(-)$$

$$\frac{\sigma_2 M_2, \sigma_2\Gamma \vdash_{\underline{u}_2,P} \quad FAx(p)}{p(\underline{a}_3), \Gamma \vdash_P}(-) \qquad \frac{FAx(p)}{p(\underline{a}_4), \Gamma \vdash}(-)$$

The dependence on a₁₉ a₂, 03, *a*^ destroys the *regularity* property (2.2), since different forms of proof trees may correspond to the same axiom/rule occurrences: hence they are similar, though uncomparable, pt's.

Finally, to every normal program *P* we associate a set *Comp*(P)* of *completion* ***axioms:***

- *Success axioms.* For every clause of P, the corresponding success axiom:
  ***SAx(A : -L_u ...,!„) =_def V(L_X* A - A L_n ^ A )**
- *Failure axioms.* For every predicate symbol p(x), the corresponding failure axiom:
  ***FAx(p) =_def* V(p(x)_ -> *3u_x(x_= t_x_*A Mi) V • • • V_*3uk(x =t_kA M_k))***
  If *k* = 0 (i.e. no clause contains *p{x)* in the head), *FAx(p)* is Vx.-»p(x) and the corresponding *AAR* is the obvious one.

A *SLDNF-proof* tree II with *axioms* from *Comp*(P)* is a pt which uses the rules (+), (-) to apply the axioms of *Comp*(P)* and restricted (s) and (w) to apply the corresponding logical axioms. We say that a goal is an *assumption* of a *SLDNF-pvoof* tree, if it occurs in some leaf. *SLDNF-proofs* will be *SLDNF*-proof trees without assumptions.

The set of $SLDNF$-proof trees is an example of what we will call a *non-regular search space*, since property (2.2) does not hold. Before better analysing non-regularity, we relate $SLDNF$-proofs to standard $SLDNF$ resolution. Let us consider finitely failed $SLDNF$-trees and $SLDNF$-refutations with a safe selection function (only ground negative literals are selected), as defined in [Llo87]. We claim the following result.

**Theorem 1** *Let $P$ be a normal program and $L$ a literal. If there is a finitely failed $SLDNF$-tree for $P \cup \{\leftarrow L\}$, then there is a $SLDNF$-proof $\Pi$ with axioms from $Comp^*(P)$ which is a continuation of $L \vdash$. If there is a $SLDNF$-refutation of $P \cup \{\leftarrow L\}$, then there is a $SLDNF$-proof $\Pi$, with axioms from $Comp^*(P)$, which is a continuation of $\vdash L$.*

**Proof.** By induction on the rank of the finitely failed $SLDNF$-tree or of the $SLDNF$-refutation.

- *Basis for finitely failed trees.* Let $\pi$ be the tree on the left below for the goal $\leftarrow A$, matching with unifiers $\sigma_{i_1}, \ldots, \sigma_{i_n}$ the clauses $B_{i_1} \leftarrow M_{i_1}, \ldots, B_{i_n} \leftarrow M_{i_n}$, and then finitely failing as suggested by the dots. The corresponding SLDNF-proof is shown on the right. Note that every continuation applies the failure axiom of the selected atom and that all the leaves are axioms (i.e. there is no assumption and our pt is a proof).

$$\begin{tikzcd}[column sep=small] & \leftarrow A \arrow[dl, no head] \arrow[dr, no head] & \\ \leftarrow \sigma_{i_1} M_{i_1} \arrow[d, phantom, "\cdots\cdots"] & \cdots & \leftarrow \sigma_{i_n} M_{i_n} \arrow[d, phantom, "\cdots\cdots"] \\ {} & & {} \end{tikzcd} \qquad \dfrac{\begin{matrix} \cdots & & \cdots & \\ \sigma_{i_1} M_{i_1} \vdash & \cdots & \sigma_{i_n} M_{i_n} \vdash, & FAx(p) \end{matrix}}{A \vdash}(\text{-})$$

  Remark that no substitution on the root of the proof tree arises from this construction, i.e. the root remains the starting goal $A \vdash$.

- *Basis for refutations.* Let $G_0, G_1, \ldots, G_n$, (with clauses $C_1, \ldots, C_n$ and substitutions $\theta_1, \ldots, \theta_n$) be a refutation, with $G_0 = \leftarrow L$. Starting with $i = 0$, associate to $G_0, \ldots, G_i$ (where $G_i = \{L_{i_1}, \ldots, L_{i_{k_i}}\}$) a pt with assumptions $\vdash L_{i_1}, \ldots, \vdash L_{i_{k_i}}$, as follows:

  *Step* 0. Associate $\vdash L$ to $G_0$;

  *Step* $i + 1$. Let $\Pi :: \vdash \delta_i L$ be the pt associated to $G_0, \ldots, G_i$ at *step* $i$, and let $G_{i+1}$ be obtained applying $C_i$ to the selected atom $A$; build the canonical continuation of the pt, selecting the corresponding assumption $\vdash A$ and applying $SAx(C_i)$ (for every step $k$, $\delta_k = \theta_1 \cdots \theta_k$ and the assumptions of the pt correspond to $G_k$). Remark that the goal proved by the final pt is the instance by $\delta_n$ of the starting goal.

- *Step for finitely failed trees.* Let $\pi$ be a finitely failed $SLDNF$-tree of rank $k + 1$, with root $\leftarrow L$. Starting from the root of $\pi$, for every node where the selected literal $L_m$ is positive, translate it as in the basis. If $L_m = \neg A$, we may have the following cases.

1. We are in a leaf of TT, $L_m$ = ->>A (with *A* ground) and there is a *SLDNF*-refutation of rank *k* of *P* U {<— A}. By inductive hypothesis, we have a *SLDNF-proof* II of h A (A is ground and no substitution modifies it). Build the continuation replacing the goal containing the selected literal by the proof:

$$\frac{\begin{array}{l}\mathbf{n}\\ \text{HA}, \quad \text{V(^i-»^A)}\end{array}}{\neg A, \Gamma \vdash}\,(\mathbf{s})$$

2. We axe in an intermediate node of TT (there is a finitely failed tree for *P* U {<— A}, hence there is no refutation of rank *k* for *P* U {<— A}). Build the continuation:

$$\frac{\text{T h}, \quad \text{V(-A -> irue)}\ ,^{x}}{\neg A, \Gamma \vdash}\,(\text{w})$$

- *Step for refutations.* We proceed as in the basis, by *Step* 0 and *Step i* + 1, provided that the selected literal $L_m$ is positive. But in *Step i + 1* we may have $i_m$ = -iA. In this case, there is a finitely failed tree of rank *k* for *P* U {<— A}. By inductive hypothesis, we have a *SLDNF-pvooi U :: A* k Then we can build the proof:

  $$\frac{\begin{array}{l}\Pi\\ A \vdash, \quad \forall(\neg A \rightarrow \neg A)\end{array}}{\text{h - A}}\,(\mathbf{s})$$

  and replace with this proof the assumption h ->>A of the pt built at step ***i.***

  Remark that the goal proved by the final pt may be more instantiated than the starting goal only if at least a positive literal has been selected.

●

*SLDNF*-resolution works in an unsafe way if open negative goals axe selected. In our model, we can distinguish two different causes for that:

(a) *soundness* problems (substitutions on existential parameters)

(b) *completeness* problems (non regularity of the search space).

As far as (a) is concerned, this corresponds to the fact that a substitution on the existential parameters (eigenvariables) may be introduced in some continuation step, as shown by the following example (taken from [Llo87], pp. 93-94).

***Axl p <— -^q(X)***

***Ax2 q(a)***

In standard Prolog, using an unsound selection function, the goal <— -ip succeeds (since •— *p* fails), although it is not a logical consequence of the completion of the program. In our model, safeness is enforced not by an external condition on the selection function, but by the usual proof-theoretic proviso on existential parameters, i.e., that they cannot be instantiated by substitutions, as the following pt shows. Once we have obtained the goal $\vdash q(u)$, with existential parameter $u$, we cannot continue our proof tree in a sound way; so we do not obtain any proof of $\neg p$. This shows that we have a natural way to distinguish proofs of negated goals (where such a proof has no assumptions) from proof trees with assumptions that cannot be continued. The latter corresponds to unprovability.

$$\dfrac{\dfrac{\dfrac{\vdash_u q(u)}{\neg q(u) \vdash_u}\,(\mathrm{s}) \quad p \to \exists x \neg q(x)}{p \vdash}\,(-)}{\vdash \neg p}\,(\mathrm{s})$$

With respect to (b), we show that there are $SLDNF$-proofs which are not achievable by usual $SLDNF$-resolution, namely that the search strategy is *incomplete* w.r.t. the problem of finding $SLDNF$-proofs. We show that this kind of incompleteness is due to the ***non-regularity*** of the rule (-).

Let us consider the following example.

$$\begin{array}{ll} Ax1 & even(0). \\ Ax2 & even(s(X)) :- \neg even(X) \end{array}$$

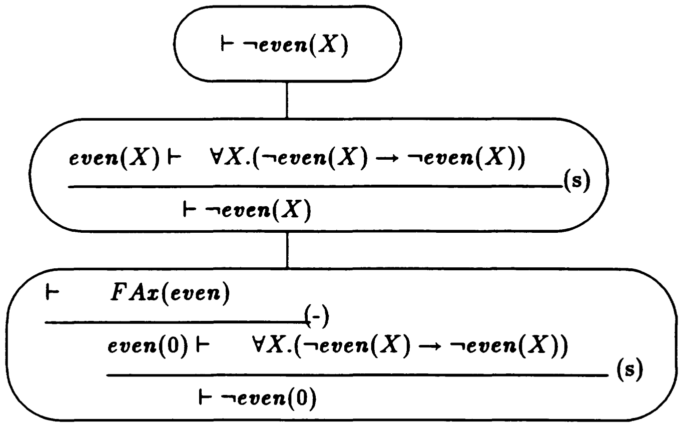
Its completion contains the obvious success axioms and the following failure axiom:

$$FAx(even) =_{def} \forall x.(even(x) \to x = 0 \vee \exists u(x = s(u) \wedge \neg even(u)))$$

and a $SLDNF$-proof of $\vdash \neg even(s(0))$ is:

$$\dfrac{\dfrac{\dfrac{\dfrac{Ax1}{\vdash even(0),}\,(+) \quad \neg even(0) \to \neg even(0)}{\neg even(0) \vdash \quad FAx(even)}\,(\mathrm{s})}{even(s(0)) \vdash \quad \neg even(s(0)) \to \neg even(s(0))}\,(\text{-})}{\vdash \neg even(s(0))}\,(\mathrm{s})$$

On the other hand, if we start from $\vdash \neg even(X)$, we obtain a finitely failed (non safe) tree. In our attitude, this tree represents the following search steps for a $SLDNF$-proof:


⊢ ¬even(X)
even(X) ⊢ ∀X.(¬even(X) → ¬even(X))
(s)
⊢ ¬even(X)
⊢ FAx(even)
(-)
even(0) ⊢ ∀X.(¬even(X) → ¬even(X))
(s)
⊢ ¬even(0)

The final proof tree is sound (no binding on existential parameters arises), but it is not a proof: we have the unprovable assumption $\vdash$. Then, we ought not interpret failure as negation: here we do not reach any proof. The situation is different from the previous ground case, where search successfully reaches a proof of $\vdash \neg even(s(0))$.

As one can see, we do not arrive at a solution since no backtracking is made on substitutions: indeed, starting from $\vdash \neg even(s(X))$ we would have attained a solution. The need to backtrack on substitutions is due to the non regularity of the search space. If only ground negated goals are selected, then failure rules are regular, since no further instantiation is possible.

Thus, to get completeness w.r.t. the problem of finding $SLDNF$-proofs, backtracking on substitutions is needed. One could suspect that there are further reasons of incompleteness, namely the lack of backtracking on the rules (+), (-), (w), (s) and on the formulae in negative goals.

One easily sees that no backtracking on the selected rule is required (indeed, for every literal $L$ in a goal, only one of (+), (-), (w), (s) can be used). On the contrary, backtracking on the selected literal (in a negative goal) could be needed, due to the rule (w), which deletes a negated formula $\neg A$ from the goal, or (s), which switches it to a positive goal and deletes the others: in both the cases, the deleted literals could be the source of success.[3] But, looking at the proof of Theorem 1, one sees that the needed backtracking is implicitly performed by a $SLDNF$-interpreter: first an attempt is made to apply the rule (s) and (w) is applied only when this attempt fails.

Therefore the reason of incompleteness is the non regularity of (-). To recover regularity, we may split the failure axiom of a predicate. For example, we split $FAx(even)$ in:

$$FAx(even(0)) =_{def} even(0) \rightarrow true$$
$$FAx(even(s(X))) =_{def} \forall X.(even(s(X)) \rightarrow \neg even(X))$$

Then, by $FAx(even(0))$, our search fails, but by $FAx(even(s(X)))$ it is successful by backtracking on the axioms. This corresponds to the fact that, for searches

[3]If rules on negative goals do not delete or add formulae, one can prove an independence result on the selected formula.

only using natural numbers, the above split axioms give rise to a regular search space. Splitting failure axioms in ground instances always gives rise to regular search spaces. The problem is that, in general, there are infinitely many ground instances, i.e., we obtain a regular but infinite axiomatization. This is analogous to what is (theoretically) suggested in [Apt88], where the grotmd instantiation of the program is adopted to obtain answers to negative open queries. On the contrary a regular splitting of the failure axioms is finite, if it exists, even for programs with infinite Herbrand universe. One possibility is to identify a *covering* of the ground terms of the language, i.e. a finite set of terms such that every other term is obtained through instantiation of the elements of the covering. In the example above, *{0,s(X)}* is a covering for the natural numbers. Let us proceed one step further: consider the $\leq$ relation defined as follows

***Axl*** $\quad 0 \leq X$

***Axl*** $\quad s(X) \leq s(Y) \leftarrow X \leq Y$

A covering for pairs of numerals is {(0,X), *(s(X),s(Y)), (s(X),0)}* This induces the following regular splitting of the failure axioms:

***Axl*** $\quad 0 \leq X \rightarrow true$

***Axl*** $\quad s(X) \leq s(Y) \rightarrow X \leq Y$

***Ax3*** $\quad s(X) \leq 0 \rightarrow false$

Note that if we dispose of the redundant first axiom, this procedure could be looked at as the synthesis of the complement of the predicate defined in the original program, namely here the 'greater' relation.

***Axl*** $\quad s(X) \leq^c s(Y) \leftarrow X \leq^c Y$

***Axl*** $\quad s(X) \leq^c 0$

Hence, when the covering creates implications with *false* as consequent, they can be turned into the base case of the complement of the predicate. This is very closely related to the notion of *intensional negation* developed in [Bar90]. It is not clear yet how general this method might be.

Last we remark that, since failure rules do not introduce unifying substitutions (as shown in the proof), dangerous substitutions can arise only if a positive assumption h *A* is selected in a continuation step, and the related unification modifies the existential parameters of some failure rule; but this cannot happen (using new names when possible) if no open negated formula is selected in a continuation. This is particularly true for definite programs and open negative queries: as no switch is possible after the initial one, no soundness problem can arise. Moreover, if we control that no link is made on existential parameters, then we may select open negated goals, as it is well known.

## 5 Related Work

In this section we review (some) papers related to our approach.

## 5.1 Resolution

- Gallier [Gal86] gives a general presentation of resolution theorem proving as derivation in certain sequent calculi: the given set of clauses is presented as a sequent where every cedent is exclusively composed of atoms: the goal is to derive the empty sequent by means of different versions of the cut rule, to which various refinements of resolution correspond. The starting point is CNF: every such clause $\neg A_1 \vee \cdots \vee \neg A_n \vee B_1 \vee \cdots \vee B_m$ can be put in the so-called Kowalski's normal form $A_1 \wedge \cdots \wedge A_n \rightarrow B_1 \vee \cdots \vee B_m$. This looks like a sequent. Then just take clauses as sequent axioms and try to derive the empty sequent, alias the empty clause. The only relevant rule (apart from factoring) is a cut with unification, where $\Gamma, \Delta$ are possibly empty sets of atomic formulae and $\theta = mgu(A, A')$.

  $$\frac{\Gamma \vdash \Delta, A \quad A', \Gamma' \vdash \Delta'}{\theta(\Gamma, \Gamma' \vdash \Delta, \Delta')}(\theta - cut)$$

  From this perspective, logic programming is linear resolution on definite clause with selection function. It is easy to enforce it by syntax, where the first premise is the program clause and the second the goal.

  $$\frac{\Gamma \vdash A \quad A', \Delta \vdash}{\theta(\Gamma, \Delta \vdash)}(sld - cut)$$

  More recently Snyder [Sny91] has embedded these systems in a two-sorted paramodulation calculus, in order to cope with equality, and the only rules are those for equality – namely, identity, left and right paramodulation.

- The *Simplified Problem Reduction Format* [Pla88] is a Gentzen system that employs sequents of the form $\Gamma \vdash A$ to perform theorem proving in the non Horn fragment. Inferences rules are generated from the input set of clauses as follows. Given a Horn clause $H \leftarrow H_1, \ldots, H_n$ we associate the inference rule

  $$\frac{\Gamma \vdash H_1 \cdots \Gamma \vdash H_n}{\Gamma \vdash H}(\mathrm{H})$$

  For each non-Horn clause $H_1, \cdots, H_m \leftarrow L_1, \ldots, L_n$, there is the following *splitting* rule

  $$\frac{\Gamma \vdash L_1 \cdots \Gamma \vdash L_n \quad \Gamma, H_1 \vdash U \cdots \Gamma, H_m \vdash U}{\Gamma \vdash U}(\mathrm{split})$$

  In a refutational setting U can be taken as to be empty, and a set of clause is unsatisfable if the empty sequent $\vdash$ can be derived. It is easy to show that the two rules are derivable in Gallier's calculus through a sequence of cuts and contractions.

- Fitting [Fit90] among other things, stresses the similarities between resolution and tableaux systems, where the usual operation of semantic tableaux a' la Smullyan axe seen as skolemization steps until resolutions steps are applicable, i. e. the atomic level has been reached for some node. The beauty of Fitting's book lies in the demonstration of the essential parenthood that links all those calculi, at the price of a somehow unfamiliar formulation of them.

## 5.2 Logic Programming

The first step is to view, through simple classical equivalencies, Horn clauses positively as rules and goals as existentially closed conjimctions of atoms to be proved by the former. Historically this can probably be dated back to Gabbay and Reyle [Gab84]. As mentioned before, we can distinguish among two approaches:

1. Clauses as axioms (programs as theories) and some form of Gentzen sequent calculus to infer goals, as it is very clearly expressed in a series of papers by Miller et al. [Mil91]. This approach is also motivated by the enlarged language in consideration (hereditary Harrop formulae), where every connective and quantifier is allowed by the syntax, though not arbitrarily.

2. Clauses as rules [Hal90]: Horn (and beyond) programs should be seen as set of inference rules for the derivation of (not necessarily ground) atoms. This view is coherent with the idea of programs defining a continuos mapping on the lattice of Herbrand interpretations: then logic programs can be seen as inductive definitions of such interpretations. A formal system *C(P)* is associated to a program *P* consisting of rules somehow similarly to what has been done above. Differently from us, they define a specialized calculus, called *linear derivation,* which proves pairs of the form (C?, 0), and demonstrate it to be sound and complete w.r.t. *C(P).* A linear derivation is essentially a SZr-D-derivation upside-down, in a structure-sharing, forward chaining style, i.e. where substitutions are split from goals. This intermediate calculus is required to actually compute the answer substitution. The rest of the paper is dedicated to enlarge the paradigm of logic programming with the notion of higher-order rules; this is connected with the possibility of having implications *as* goals, which is also a feature of Miller's approach. This is also the basis of the sequel of the paper, where [Hal91] a definational approach to logic programming is sketched:

W.r.t. negation-as-failure Stärk [St92a] has (independently) given a sequent formulation of Clark's completion that is very close to ours. His calculus *NF(P)* consists of

- Clark's equality and freeness axioms
- Negation rules (our switch rules)
- cut rules, where S is a set of equations and F of literals.

$$\overline{\Sigma, \Gamma \vdash A} \qquad \overline{\Sigma; \Gamma \vdash}$$

- Program rules from *P*, divided into positive and negative introduction. For example, given the above program for *even* we have:

$$\frac{\Sigma \vdash t = 0}{\Sigma \vdash even(t)} \qquad \frac{\Sigma \vdash t = s(X) \quad X \vdash^{-} even(X)}{\Sigma \vdash even(t)}$$

$$\frac{t = 0, \Gamma \vdash \quad t = s(X), \neg even(X), \Gamma \vdash}{even(t), \Gamma \vdash}$$

Much more is however contained in Stärk's thesis; to quote a few, he shows that a sequent is provable in *NF(P)* iff it is true in all 3-valued model of the completion. Furthermore a completeness result is proved w.r.t *SLDNF*- resolution for program satisfying the cut-property [St92b].

Harland [Har92] proposes a sequent calculus based on intuitionisic logic that directly incorporates *NF* without referring to the completion. The usual rules, restricted to deal with Horn logic, induce a *positive* derivability relation; furthermore, he introduces a *negative* system of *disprovability,* denoted by $\vdash^{-}$ with judgments of the type: if a conjunction fails then so does one of the conjuncts and so forth. The two systems are interlinked by the rules for *NF,* which corresponds to our switch rules (s+) (s-).

$$\frac{\Gamma \vdash^{-} F}{\Gamma \vdash \neg F}(\neg - R+) \qquad \frac{\Gamma \vdash F}{\Gamma \vdash^{-} \neg F}(\neg - R-)$$

Structural rules have a paramount importance in this calculus: due to the nonmonotonic nature of *NF,* in the positive fragment weakening is restricted to introduce only already provable formulae on the left, resulting to be dual to contraction. In the negative system structural rules, cut included, are not eliminable.

Remark that this is not stricltly a calculus *of finite* failure, since sequents of the form $p \leftarrow p \vdash^{-} p$ are provable, although with the essential use of weakening. Of course, due to the recursion-theoretic complexity of unprovability, every such an axiomatic formulation is bound to be incomplete. Moreover the boundary seems fuzzy, as, for example, *p* is independent from the program $p \leftarrow q, q \leftarrow p$.

The calculus without structural rules is shown [Har91] to be sound and complete w.r.t. finite failure and SLDNF-resolution (extended to the first-oder hereditary Harrop formulae). Having to model the operational behavior of *NF,* both the positive and the negative system make explicit references to unification, Skolem functions and so forth. Still, the system is not meant for proof search and requires a relative complement algorithm to provide answers to open negative queries.

## 6 Conclusion and Future Work

Concluding, we believe that our proof-theoretic interpretation gives a clear explanation of some well known phenomena and suggests some interesting research directions. From our original goal of giving a proof-theoretic reconstruction of logic programming, as started in [Orn92], we are now in the position of proposing a significant enrichment of the logic programming paradigm. This alone is a good evidence of the fruitfulness of the this paradigm.

Moreover, this parabola is been common to other researchers: by stressing the constructive features of logic programming [Mil91] has formulated the notion of uniform proof. Similarly, [Hal90], [Hal91] has devised a definational approach to logic programming.

Analogously, from our analysis of SZrD(JV-F)-resolution, we have started to elicit the properties (regularity, closure under substitutions et al.) that can turn any axiom-application rules system in a Prolog-like programming language. We axe developing our work towards a general theory of regular AAR's system. This will accomplished in a following companion report.

## References

[Apt88] Apt K.A. , Blair H. A. &: Walker A. Towards a Theory of Declarative Knowledge, in: *Foundations of Deductive Databases and Logic Programming.* Minker J. (ed.) Morgan Kaufmann 1988.

[Apt90] Apt K.A. Logic Programming, *Handbook of Theoretical Computer Science,* (Leuween J. ed.), Elsevier 1990.

[Bar90] Barbuti R. , Mancarella P. , Pedreschi D. & Turini F. A Transformational Approach to Negation in Logic Programming, *Journal of Logic Programming,* pp. 201-228, 8, 1990.

[Cla78] Clark K.L. Negation as Failure in: Gallaire & Minker (eds.), *Logic and Data Bases,* Plenum Press, New York, 1978.

[Fit90] Fitting M. *First-Order Logic and Automated Theorem Proving,* Springer-Verlag, 1990.

[Gab84] Gabbay D. M. &; Reyle U. N-Prolog: an Extension of Prolog with Hypothetical Implications: 1 *Journal of Logic Programming,* 4, pp. 319-355, 1984.

[Gal86] Gallier J. *Logic for Computer Science, Foundations of Automatic Theorem Proving,* Harper & Row, New York, 1986.

[Hal90] Hallnäss L. *k,* Schroeder-Heister P. A Proof Theoretic Approach to Logic Programming: Clauses as Rules, *Journal of Logic and Computation,* vl no. 2. pp. 261-283, 1990.

[Hal91] Hallnäss L. *Sz* Schroeder-Heister P. A Proof Theoretic Approach to Logic Programming: Programs as Definitions *Journal of Logic and Computation,* v.l no. 5. pp. 635-660, 1991.

[Har91] Harland J. *On Hereditary Harrop Formulae as a Basis for Logic Programming,* PhD Thesis, Edinburgh 1991.

[Har92] Harland J. Towards a Static Proof System for Negation as Failure. *Citri/TR-92-49*, University of Melburne, 1992.

[Kun87] Kunen K. Negation in Logic Programming. *Journal of Logic Programming*, 4, pp. 289-308, 1987.

[Llo87] Lloyd J.W. *Foundations of Logic Programming*, Second Extended Edition, Springer-Verlag, Berlin, 1987.

[Mil89] Miller D. A Logical Analysis of Modules in Logic Programming, *Journal of Logic Programming*, pp. 79-108, 1989.

[Mil91] Miller D. , Nadathur G. , Pfenning F. , Scedrov A. Uniform Proofs as a Foundation for Logic Programming, *The Annals of Pure and Applied Logic*, v.51, pp. 125-157, 1991.

[Orn92] Ornaghi M. & Momigliano A. A Proof-Theoretic Reconstruction of Logic Programming, (Abstract), F. Pfenning (ed.), *Workshop on Proof and Types*, JICSLP92, Washington 1992.

[Pla88] Plaisted D.A. Non Horn Logic Programming without Contrappositives, *JAR*, 4 pp. 287-325, 1988.

[She88] Shepherdson J.C. Negation in Logic Programming in: *Foundations of Deductive Databases and Logic Programming*, Minker J. (ed.), Morgan Kaufmann, 1988.

[Sny91] Snyder W & Linch C. Goal-Oriented Strategies for Paramodulation, *RTA* R.Book (ed.), Springer-Verlag LNCS, vol. 488, 1991.

[St92a] Stärk R. *The Proof-Theory of Logic Programs with Negation.* PhD Thesis, University of Bern, 1992.

[St92b] Stärk R. Cut-Property and Negation as Failure, *Technical Report*, University of Bern, 1992.