Regular Search Spaces as a Foundation of Logic Programming

Alberto Momigliano and Mario Ornaghi

Department of Philosophy, Carnegie Mellon University
15213 Pittsburgh PA, USA — mobile@lcl.cmu.edu
Dipartimento di Scienze dell'Informazione, Università degli studi di Milano
Via Comelico 39/41, Milano, Italy — ornaghi@imiucca.csi.unimi.it

Abstract. First, we aim to provide a proof-theoretic reconstruction of logic programming, both for definite and for normal programs. This leads us to a better understanding of negation-as-failure (NF) [5] and of other proof-theoretically well-founded extensions of Prolog. Our ambition is to show that almost everything in the vulgate of logic programming can be carried out in a style close to natural deduction, and in particular in a restricted fragment of minimal logic. Secondly, we embark on a generalization and abstraction of the nice properties of SLD-resolution. The outcome is our formulation of the concept of *regular search space*. Here many logics can be expressed and provided they can be shown to be *regular*, they are then guaranteed to enjoy an analogue of the very features that make Prolog a feasible and successful implementation of logic. This can also serve as a basis of a fairly general new logic programming language, similarly to what is done in [11].

1 Introduction

Most of the papers on logic programming, in particular on its extension, have been carried out in a semantic way. Unfortunately the semantics, being mainly limited to term models, tends to hide proof-theoretic contents. Moreover, the procedural aspects are traditionally expressed in a refutational style. We prefer a more direct approach that bypasses, where possible and meaningful, this pseudo-semantic/refutational style. Many features that are rather cluttered in this framework now appear as natural consequences of a proof-theoretic reading.

Basically two proof-theoretic approaches have been pursued in the literature, as outlined in [10] – see section 6 for a more detailed discussion:

1. Clauses as axioms and some Gentzen calculus to infer goals [20].
2. Clauses as rules [10]: programs should be seen as sets 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; θ is a computed answer substitution for $P \cup \{\leftarrow G\}$ iff there exists a mgpt for θG with axioms from P, with no open assumption. Mgpts are SLD-derivations

upside-down, where the root is the goal to be proven, the assumptions are the intermediate goals therein and θ is the restriction to the free variables of G of the composition of mgus along the branch. Mgpts are based on the notion of *axiom application rule* (AAR), which easily generalizes to more complex definitions of clauses and goals. Then we formulate the concept of *regular search space* for which search strategies like Prolog's are complete.

Introducing AAR 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 arises from 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 simplied synthesis technique in a spirit close to [4].

In the last part we start an abstract analysis of general AAR systems, satisfying very simple properties like regularity and closure under substitution.

The paper 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 and Prolog computations: a soundness and completeness theorem is 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. Section 5 presents the beginning of a general theory of AAR systems, with some examples. Eventually in (6) we conclude with a review of the other existing proof-theoretic studies and with a list of 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 1. *sld* is a rule which applies Horn axioms $\forall(A_1 \wedge \cdots \wedge A_n \rightarrow B)$ to goals θB, 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:

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

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* $\theta A_1, \ldots, \theta A_n$ represents the new goals and the *conclusion* θB 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. An atom A is a *proof tree*. If $\Pi_1 :: \theta A_1, \ldots, \Pi_n :: \theta A_n$ are proof trees, then the following is also a *proof tree*:

$$\frac{\begin{array}{cccc} \Pi_1 & & \Pi_n & \\ \theta A_1 & \cdots & \theta A_n & \forall(A_1 \wedge \cdots \wedge A_n \rightarrow B) \end{array}}{\theta B}$$

where $\Pi :: A$ 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 are the ones appearing as major premises. A proof tree is a *proof* of B from a set $\mathcal{A}$ of axioms iff B is its consequence, its axioms belong to $\mathcal{A}$ and it has no assumptions.

The set of proof trees is closed under substitution, namely, if Π is a proof tree and θ is a substitution, then $\theta(\Pi)$ is a proof tree. This allows us to introduce the following *pre-ordering* (intuitively to be read as Π_1 is less general or more instantiated than Π_2) and *equivalence* relation among proof trees:

Definition 3.

- $\Pi_1 \leq \Pi_2$ iff there is a θ such that $\Pi_1 = \theta(\Pi_2)$;
- $\Pi_1 \equiv \Pi_2$ iff $\Pi_1 \leq \Pi_2$ and $\Pi_2 \leq \Pi_1$;
- $Cone(\Pi) = \{\Pi' \mid \Pi' \leq \Pi\}$.

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 are 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 may involve different substitutions. Similarity is crucial, because axiom-application sequences represent derivations of an idealized interpreter which searches for proofs $\Pi :: \theta C$ 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-like search strategies complete. The formal definition of similarity is the following.

Definition 4. An *axiom-occurrence* in a pt Π is a pair $\langle p, Ax \rangle$ such that p is a path from the root to a node containing Ax as a major premise. We say that two proof trees Π_1, Π_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-occurrences.

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 Π.

Proposition 5. *$\Pi_1 \sim \Pi_2$ iff there is a proof tree Π 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 Π_1, Π_2.
Basis. There are 0 axiom occurrences. In this case, Π_1 and Π_2 are two atoms containing 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 below:

$$\Pi_1 = \dfrac{\dfrac{\theta_1 A_1 \cdots \quad \theta_1 A_n \quad Ax}{\theta_1 B}}{\Gamma_1} \qquad \Pi_2 = \dfrac{\dfrac{\theta_2 A_1 \cdots \quad \theta_2 A_n \quad Ax}{\theta_2 B}}{\Gamma_2}$$

Γ_1 is similar to Γ_2, since both have axiom occurrences $\{o_1, \ldots, o_{n+1}\} - \{o_k\}$; by inductive hypothesis, there is a proof tree Γ such that $\Gamma_1 = \sigma_1\Gamma$ and $\Gamma_2 = \sigma_2\Gamma$. Let H be the top formula occurring in Γ, in the place of $\theta_1 B$ in Γ_1 (and of $\theta_2 B$ in Γ_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 δ 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 μ_1, μ_2. Then we can build the pt Π:

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

Π is similar to Π_1 and Π_2, and one easily sees that $\Pi_1 = \mu_1\Pi$ and $\Pi_2 = \mu_2\Pi$ (we can assume that the variables of Γ 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 the comparability. This does not hold w.r.t. a more general kind of AAR, e.g. the ones related to negation as failure, as shown in section 4.

Proposition 6. *For every set $\mathcal{S}$ of similar proof trees, there is a Π such that $\mathcal{S} \subseteq Cone(\Pi)$.*

Proof. Assume the contrary, i.e. that for every Π there is a Π^* in $\mathcal{S}$ such that $\Pi^* \not\leq \Pi$ and let Π_0 be a pt in the similarity class containing $\mathcal{S}$: there is a pt Π_1 in $\mathcal{S}$ such that $\Pi_1 \not\leq \Pi_0$. By (5) they have a similar upper bound, say Δ_1, such that $\Pi_0 = \theta_0\Delta_1$. If we define $m(\Pi)$ as the sum of the sizes of the terms occurring in a pt Π, then $m(\Pi_0) > m(\Delta_1)$ since θ_0 cannot be a renaming. Consider $\Pi_2 \not\leq \Delta_1$:

by the same reasoning, they are bounded by Δ_2. Iterating the process we create an infinite chain $\Pi_0, \Delta_1, \Delta_2, \ldots$, where $\Pi_0 = \theta_0\Delta_1$, $\Delta_1 = \theta_1\Delta_2, \ldots$, but this is not possible since $m(\Pi_0) > m(\Delta_1) > m(\Delta_2) > \ldots$.

Remark. For every Π, $Cone(\Pi) \subseteq Sim(\Pi)$.

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

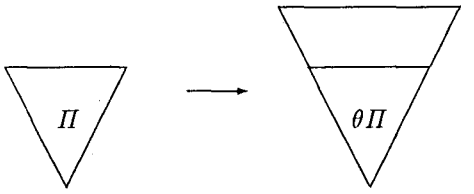
Definition 7. We say that Π is a *most general proof tree* (mgpt) if it is a *maximal* element among similar trees, i.e. $Cone(\Pi) = Sim(\Pi)$.

By (5), every similarity class $Sim(\Pi)$ contains a maximal element unique up to renaming. On the other hand minimal elements among similar trees are ground proof trees, but in general there are 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 8. Π_2 is a *continuation* of Π_1, denoted $\Pi_1 \preceq \Pi_2$, iff there is an initial subtree Π_3 of Π_2, that is with the same root, s.t. $\Pi_3 \leq \Pi_1$.

Assuming that the root is the goal to be proved, a continuation is a possible (not necessarily single-step) extension of a SLD-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 Π is continued into a proof tree containing $\theta\Pi$ as an initial subtree.



The $\preceq$ 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 feature of logic programming. This characteristic is better represented by the notion of most general continuation and canonical continuation and by their properties that we list below.

Remark. $\Pi' \leq \Pi$ entails $\Pi \preceq \Pi'$.

Definition 9. Let Π be a pt and $\Pi \preceq \Pi^*$. We say that Π^* is a *most general continuation* (*mgc*) of Π iff, for every continuation Π' of Π similar to Π^*, $\Pi' \leq \Pi^*$.

Proposition 10. *For every continuation Π' of Π, there is a mgc Π^* of Π such that $\Pi' \leq \Pi^*$.*

Definition 11. A *one-step* continuation *selecting* an assumption H_i and *applying* an axiom Ax of the form $\forall(A_1 \wedge \cdots \wedge A_k \rightarrow B)$ s.t. $\sigma B = \sigma H_i$ is of the following form:

$$\cdots \quad \frac{\sigma A_1 \cdots \sigma A_k \quad Ax}{\begin{array}{c}\sigma H_i \\ \sigma \Pi\end{array}} \quad \cdots$$

We say that a continuation is *canonical* iff it is a one-step continuation and σ is a most general unifier of the selected assumption and the head of the applied axiom (assuming the usual standardization apart).

The following relations hold between subsumption and continuation, embodying the reason why search may be conducted exclusively on mgcs or on mgpts and no backtracking on substitutions is required, as we will see in the next section.

Proposition 12. *If $\Pi_1 \leq \Pi_2$ and $\Pi_1 \preceq \Pi_3$, then $\Pi_2 \preceq \Pi_3$.*

Proposition 13. *Let Π' be a one-step continuation of Π. Π' is a mgc of Π iff it is a canonical continuation.*

Proposition 14. *If Π is a mgpt, then its mgcs are mgpts. In particular, its canonical continuations are mgpts.*

We conclude this section considering the consequences of the above properties with respect to the following search problem. Let $\mathcal{A}$ be a set of axioms, $T(\mathcal{A})$ the set of the pts with axioms from $\mathcal{A}$, and $\Pi :: C$ be a pt of $T(\mathcal{A})$. Search for a continuation $\Pi^* :: \theta C \in T(\mathcal{A})$ of Π, such that Π^* is a proof. One easily sees that the notions of mgpt, mgc and all the related properties hold even though we consider $T(\mathcal{A})$, instead of the class of all proof trees. Then one obtains:

Corollary 15. *If Π_1 is a mgpt of $T(\mathcal{A})$, then, for every Π_2 such that $\Pi_1 \sim \Pi_2$, if there is a Π s.t. $\Pi_2 \preceq \Pi$, then $\Pi_1 \preceq \Pi$.*

Proof. Since Π_1 is a mgpt, then $\Pi_2 \leq \Pi_1$. Apply (12).

Proposition 16. *Let Π be a mgpt of $T(\mathcal{A})$ and H_i be an assumption of Π. If for every canonical continuation Π' of Π selecting H_i there is no proof in $T(\mathcal{A})$ continuing Π', then there is no proof in $T(\mathcal{A})$ continuing Π.*

Proof. Suppose the contrary, that there is a pt Π^* continuation of Π. Then there is a subpt $\hat{\Pi}$ of Π^* that is a one-step continuation of Π selecting H_i. Hence $\hat{\Pi}$ is similar to Π'. But, by (14), Π' is a mgpt and, by (15), Π^* is a continuation of Π'.

Corollary 15 more properly refines what we mean by "avoiding backtracking on substitutions". (16) shows that, by using canonical (i.e. most general) continuations, during the search the selection of the assumption H_i may be completely non-deterministic. (15) and (16) allow us also to state that a search strategy like the one of Prolog is complete in the search spaces for $T(\mathcal{A})$. But (15) and (16) are based on (5), which may not hold in an arbitrary search space. When the latter is the case, we will say that the *regularity property* holds. This remark will be the basis of section 5, where we discuss *regular search spaces*, i.e. search spaces satisfying the regularity property 5, for a very general kind of axioms. In the 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:

$$sum(X, 0, X).$$
$$sum(X, s(Y), s(Z)) : -sum(X, Y, Z)$$

and the corresponding axioms $Ax(SUM)$

$$Ax1 \quad \forall X.sum(X, 0, X)$$
$$Ax2 \quad \forall X, Y, Z.(sum(X, Y, Z) \rightarrow sum(X, s(Y), s(Z)))$$

Examples of proof trees Π_0, Π_1 and Π_2 using $Ax(SUM)$ are:

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

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

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

Π_0 has assumption $sum(X, 0, s(0))$ and consequence $sum(X, s(s(0)), s(s(s(0))))$; Π_1 and Π_2 are proofs (of their consequences). Π_1 and Π_2 are similar, since they have the same set of axiom occurrences $\{\langle[1], Ax2\rangle, \langle[0,1], Ax2\rangle, \langle[0,0,0], Ax1\rangle\}$ (paths are represented by sequences of numbers, in the usual way); the following proof tree is the mgpt in their similarity class:

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

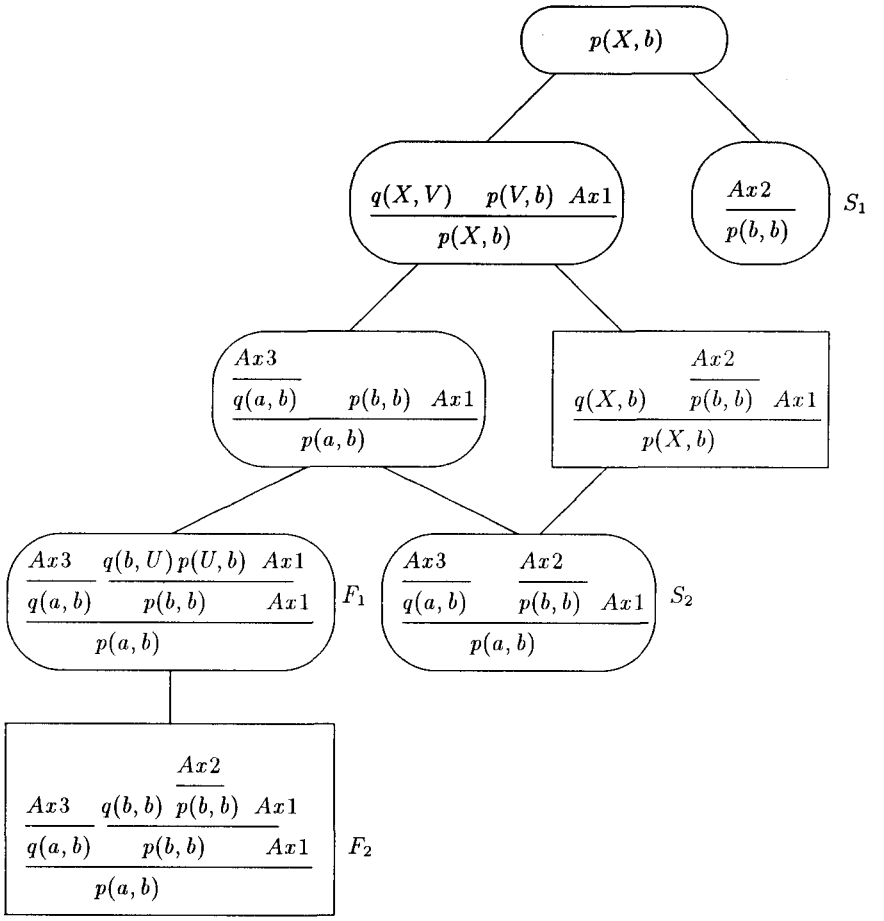
Now we address the following problem: given a program P and an atomic formula C, search for a proof $\Pi :: \theta C$. From now on, in this section, we will call C a *goal*. We give the following definition.

Definition 17. 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 mgpts.

Example 1. Let us consider the following program P[1] (see [17], 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 $p(X,b)$:



The success nodes S_1, S_2 contains *proofs* and the proof tree in the failure node

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

F_2 has open assumptions, but no continuations. The oval boxes show the canonical continuations obtained by choosing the leftmost assumption. The rectangles show the canonical continuations selecting other assumptions. By (16), we obtain, selecting other assumptions, more proof trees, but the success set remains the same. Then we can stop at F_1, taking it as a failure node.

As last step consider the quotient $Gen(P, C)/\equiv$ under renaming of variables. This passage is required since each resolution step considers variants of the input clauses to ease unification. We claim that $\langle Gen(P, C)/\equiv, \preceq\rangle$ corresponds to every SLD-tree for $P \cup \{\leftarrow C\}$ under each selection rule, 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 $[\Pi :: \theta C]$, with $\Pi :: \theta C \in Gen(P, C)$
- every child of a node is obtained by a canonical continuation applying an axiom of $Ax(P)$.

Proposition 18. *θ is a computed answer substitution for $P \cup \{\leftarrow C\}$ iff there is a $\Pi :: \theta C \in Gen(P, C)/\equiv$, with no open assumption.*

Proof. Two straightforward inductions, respectively on the length of the SLD-refutation and on the structure 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 [17] theorem 9.2. This result establishes a sort of Church-Rosser property for SLD-resolution; if $P \cup \{\leftarrow C\}$ 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 16 of regular search spaces[2]. 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 Π a fixed assumption in a leaf, called the *assumption selected by* F and denoted by $F(\Pi)$. We require that $F(\Pi)$ depends only on the rules applied in Π, i.e. that for any similar Π, Π', $F(\Pi)$ and $F(\Pi')$ select the same leaf.

Definition 19. Let F be a selection function. A *F-proof-sequence* is a (possibly infinite) sequence $[\Pi_0], [\Pi_1], \ldots, [\Pi_n], \ldots$ such that (for $i \geq 0$) Π_{i+1} is a canonical continuation of Π_i selecting the assumption $F(\Pi_i)$. An *F-search tree* for a program P and a goal C is a subtree of $\langle Gen(P, C)/\equiv, \preceq\rangle$, such that every child of a node is obtained by a canonical continuation applying an axiom of $Ax(P)$ and selecting the assumption chosen according to F.

[2] Indeed, a SLD-derivation of $P \cup \{\leftarrow C\}$ with current goal $\leftarrow H_1, \ldots H_n$ corresponds to a derivation of a proof tree $\Pi :: \theta C$ with assumptions $H_1, \ldots H_n$; apply (16) to $\Pi :: \theta C$.

The independence result says that the proofs (i.e. pts without assumptions) contained in $\langle Gen(P,C)/\equiv, \preceq\rangle$ and the ones contained in a *complete* F-search tree (namely an 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 are 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 [17] on the selection function to ensure that we find a finitely failed tree if one exists. Fairness can be characterized as follows.

Definition 20. A F-proof-sequence is *fair* if it is finite or every (instance of an) atom appearing in it is selected by F. An F-search tree is *fair* if every path of the tree is a fair F-proof-sequence.

Proposition 21. *If there is a finitely-failed fair F-search tree 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 22. *There is a finitely-failed 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 Π 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 [5], 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 rule does 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:

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

Negative Goals and the Switch and Weakening Rules. The negative goals are of the form $\Gamma \vdash$, where Γ 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):

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

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 $\vdash A$ fails (on the left hand side) and the goal fails since $\vdash A$ succeeds (on the right hand side). Note, however, that under the same proviso the rule corresponds to $\neg$-R and $\neg$-L in the sequent calculus with primitive negation. The weakening rule (w) is:

$$\frac{\Gamma \vdash \quad \forall(\neg A \rightarrow \mathit{true})}{\neg A, \Gamma \vdash}(\mathrm{w})$$

To achieve a uniform treatment, we have expressed the rules (s) and (w) as *AARs*, 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 [5]) 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* (id1),(id2) and the *substitution rules*, (u1) and (u2) which are derivable from the freeness axioms. Note that this equational theory can be shown to be complete and decidable ([15], Theorem 5.5).
  - (id1) $\vdash t = t$
  - (id2) $t_1 = t_2, L(t_1) \vdash L(t_2)$
  - (u1) $t_1 = t_2 \vdash Eq(\sigma)$ if σ is an idempotent mgu of t_1, t_2 and $Eq(\sigma)$ is the conjunction of the equations obtained from the bindings in σ
  - (u2) $t_1 = t_2 \vdash$ if no unifier of t_1, t_2 exists

  Now, consider the completed definition of a predicate p of the form

$$\forall \underline{x}(p(\underline{x}) \leftrightarrow \exists \underline{u}_1(\underline{x} = \underline{t}_1 \wedge M_1) \vee \exists \underline{u}_2(\underline{x} = \underline{t}_2 \wedge M_2))$$

Here we are assuming that p is defined by two clauses with body M_1, M_2; it is clear how to extend it to the general case. Consider the direction $\rightarrow$, that we call the *failure axiom* of p. Thanks to that, we can build proof trees of the form shown in the following, where $\underline{a}$ is a list of terms and, for the sake of this example, we suppose that σ_1 is a mgu of $\underline{a}, \underline{t}_1$ and that $\underline{a}, \underline{t}_2$ do not unify.

$$\dfrac{\dfrac{\dfrac{\dfrac{\sigma_1 M_1, \sigma_1 \Gamma \vdash_{\underline{u}_1}}{\underline{a} = \underline{t}_1 \vdash_{\underline{u}_1} Eq(\sigma_1)\ \ Eq(\sigma_1), M_1, \Gamma \vdash_{\underline{u}_1}}(*)(\text{cut})}{\underline{a} = \underline{t}_1, M_1, \Gamma \vdash_{\underline{u}_1}}(\exists L)}{\exists \underline{u}_1 (\underline{a} = \underline{t}_1 \wedge M_1), \Gamma \vdash} \quad \dfrac{\dfrac{\underline{a} = \underline{t}_2 \vdash_{\underline{u}_2}}{\underline{a} = \underline{t}_2, M_2, \Gamma \vdash_{\underline{u}_2}}(\exists L)}{\exists \underline{u}_2 (\underline{a} = \underline{t}_2 \wedge M_2), \Gamma \vdash} \quad FAx(p)}{p(\underline{a}), \Gamma \vdash}(\text{f})$$

Let $FAx(p)$ denote the failure axiom of p: (f) can be derived from $p(\underline{a}) \vdash \exists \underline{u}_1 (\underline{a} = \underline{t}_1 \wedge M_1) \vee \exists \underline{u}_2 (\underline{a} = \underline{t}_2 \wedge M_2)$. The $(\exists L)$ rules are iterated $\exists$-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 $AARs$.

– *The Negative Rule.* (-) is a derived rule whose form *depends* on the following exhaustive unification possibilities, under the guidance of substitution σ_i, given the proviso on the eigenvariables decorating $\vdash$:
$\underline{a}_1$ unifies with $\underline{t}_1$ and $\underline{t}_2$;
$\underline{a}_2$ unifies with $\underline{t}_1$ but not with $\underline{t}_2$;
$\underline{a}_3$ unifies with $\underline{t}_2$ but not with $\underline{t}_1$;
$\underline{a}_4$ does not unify with $\underline{t}_1$ nor with $\underline{t}_2$.

$$\dfrac{\sigma_1 M_1, \sigma_1 \Gamma \vdash_{\underline{u}_1 \underline{u}_2 P} \quad \sigma_2 M_2, \sigma_2 \Gamma \vdash_{\underline{u}_1 \underline{u}_2 P} \quad FAx(p)}{p(\underline{a}_1), \Gamma \vdash_P}(\text{-}) \qquad \dfrac{\sigma_1 M_1, \sigma_1 \Gamma \vdash_{\underline{u}_1 P} \quad FAx(p)}{p(\underline{a}_2), \Gamma \vdash_P}(\text{-})$$

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

By the dependence on $\underline{a}_1$, $\underline{a}_2$, $\underline{a}_3$, $\underline{a}_4$ (5) no longer holds. Indeed different forms of proof trees may correspond to the same axiom and rule occurrences:[3] hence they are similar, though uncomparable, pts.

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

[3] Enlarging the number of rules, the similarity dependencies may increase.

- *Success axioms.* For every clause of P, the corresponding success axiom:

$$SAx(A : -L_1, \ldots, L_n) =_{def} \forall(L_1 \wedge \cdots \wedge L_n \rightarrow A)$$

- *Failure axioms.* For every predicate symbol p, the corresponding failure axiom:

$$FAx(p) =_{def} \forall(p(\underline{x}) \rightarrow \exists \underline{u}_1(\underline{x} = \underline{t}_1 \wedge M_1) \vee \cdots \vee \exists \underline{u}_k(\underline{x} = \underline{t}_k \wedge M_k))$$

  If $k = 0$ (i.e. no clause contains p in the head), $FAx(p)$ is $\forall \underline{x}.\neg p(\underline{x})$ and the corresponding AAR is the obvious one.

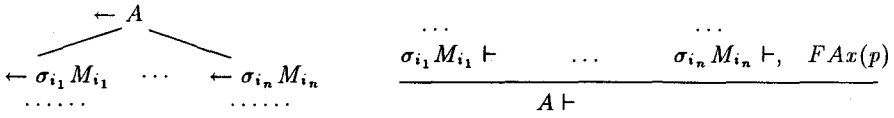
A $SLDNF$-proof tree Π with *axioms* from $Comp^*(P)$ is a pt which uses the rules (+), (-) to apply the axioms of $Comp^*(P)$ and restricted (s) and (w) to apply the corresponding logical axioms. We say that a goal is an *assumption* of a $SLDNF$-proof tree, if it occurs in some leaf. *$SLDNF$-proofs* 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 5 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 [17]. We claim the following result.

Theorem 23. *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 Π 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 Π, 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 π 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 failure axioms (i.e. there is no assumption and our pt is a proof).



  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 $G_i = \leftarrow L_{i_1}, \ldots, L_{i_{k_i}}$) 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 δ_n of the starting goal.

- *Step for finitely failed trees.* Let π be a finitely failed $SLDNF$-tree of rank $k+1$, with root $\leftarrow L$. Starting from the root of π, 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 π, $L_m = \neg A$ (with A ground) and there is a $SLDNF$-refutation of rank k of $P \cup \{\leftarrow A\}$. By inductive hypothesis, we have a $SLDNF$-proof Π of $\vdash A$ (A is ground and no substitution modifies it). Build the continuation replacing the goal containing the selected literal by the proof:

$$\dfrac{\begin{array}{ll}\Pi & \\ \vdash A & \forall(\neg A \rightarrow \neg A)\end{array}}{\neg A, \Gamma \vdash}\,(\mathrm{s})$$

  2. We are in an intermediate node of π (there is a finitely failed tree for $P \cup \{\leftarrow A\}$, hence there is no refutation of rank k for $P \cup \{\leftarrow A\}$). Build the continuation:

$$\dfrac{\Gamma \vdash \qquad \forall(\neg A \rightarrow true)}{\neg A, \Gamma \vdash}\,(\mathrm{w})$$

- *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 $L_m = \neg A$. In this case, there is a finitely failed tree of rank k for $P \cup \{\leftarrow A\}$. By inductive hypothesis, we have a $SLDNF$-proof $\Pi :: A \vdash$. Then we can build the proof:

$$\dfrac{\begin{array}{ll}\Pi & \\ A \vdash & \forall(\neg A \rightarrow \neg A)\end{array}}{\vdash \neg A}\,(\mathrm{s})$$

  and replace with this proof the assumption $\vdash \neg A$ 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 are 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 spaces).

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 [17], pp. 93-94):

$$\begin{array}{ll} Ax1 & p \leftarrow \neg q(X) \\ Ax2 & q(a) \end{array}$$

In standard Prolog, using an unsound selection function, the goal $\leftarrow \neg p$ succeeds (since $\leftarrow p$ 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 $\vdash_u q(u)$, with existential parameter u, we cannot continue our proof tree in a sound way; so we do not obtain any proof of $\neg p$.

$$\dfrac{\dfrac{\dfrac{\vdash_u q(u)}{\neg q(u) \vdash_u}\,(\mathrm{s}) \qquad p \rightarrow \exists x \neg q(x)}{p \vdash \qquad \neg p \rightarrow \neg p}\,(\text{-})}{\vdash \neg p}\,(\mathrm{s})$$

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.

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.

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

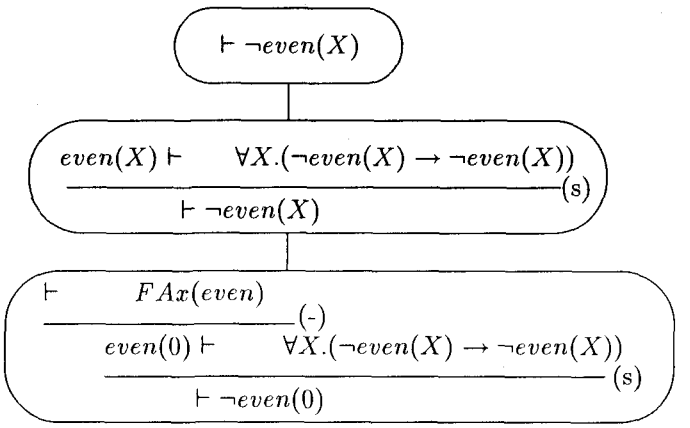
Its completion contains the obvious success axioms and the following failure axiom:

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

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

$$\dfrac{\dfrac{\dfrac{Ax1}{\vdash even(0)}(+) \quad \neg even(0) \rightarrow \neg even(0)}{\neg even(0) \vdash \quad FAx(even)}(s)}{even(s(0)) \vdash \quad \neg even(s(0)) \rightarrow \neg even(s(0))}(-)}{\vdash \neg even(s(0))}(s)$$

On the other hand, if we start from $\vdash \neg even(X)$, we obtain a finitely failed (not safe) tree. In our approach, this tree represents the following search steps for a $SLDNF$-proof:



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.

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

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 [1], where the ground 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 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

$$\begin{aligned} &Ax1 \quad 0 \leq X \\ &Ax2 \quad s(X) \leq s(Y) \leftarrow x \leq y \end{aligned}$$

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

$$\begin{aligned} &Ax1 \quad 0 \leq X \rightarrow \mathit{true} \\ &Ax2 \quad s(X) \leq s(Y) \rightarrow x \leq y \\ &Ax3 \quad s(X) \leq 0 \rightarrow \mathit{false} \end{aligned}$$

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.

$$\begin{aligned} &Ax1 \quad s(X) \leq^c s(Y) \leftarrow x \leq^c y \\ &Ax2 \quad s(X) \leq^c 0 \end{aligned}$$

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 [4]. 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 $\vdash A$ 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 Systems Based on Axiom-Application Rules

The notion of *axiom-application rule* and of *regularity* are the key points of our approach. In this section we try to offer the beginning of a general theory of *AAR*s, aiming to single out the properties that make Prolog successful.

One of the main problems of automated theorem proving is to locate proofs in a given space, known as a *search space*. For *AAR* systems, search spaces are sets of proof trees. If a proof tree Π in a search space $\mathcal{S}$ is not a proof, it represents a stage in a search process, whose aim is to find a proof $\Pi' \in \mathcal{S}$, where Π' is a continuation of Π. Typical examples are $T(P)$, the search space of a program P, and $T(P, C)$, the one of a program P with respect to a goal C, as previously defined in section 3.

Now we introduce the following general language: first we abstract from the usual notions of goal and axiom. Secondly we characterize a system of axiom application rules by giving once for all a fixed (and hopefully small) set of rules to apply axioms of a certain type. Programs are finite sets of those axioms.

5.1 *AAR* Systems

An *AAR* system is a triple $\langle \mathcal{G}, \mathcal{A}, \mathcal{R} \rangle$ where:

1. $\mathcal{R}$ is a set of axiom application rules;
2. $\mathcal{A}$ is a set of admissible axioms;
3. $\mathcal{G}$ is a set of admissible goals.

A rule $R \in \mathcal{R}$ is a *partial* function from goals and axioms to sequences of goals, including the empty sequence Λ, i.e. $R : \mathcal{G} \times \mathcal{A} \rightarrow \mathcal{G}^*$.

When $R(G, A) = G_1; \ldots; G_n$[4], we will draw it as

$$\frac{G_1; \cdots; G_n; \quad A}{G}(\mathrm{R})$$

When $R(G, A) = \Lambda$, we write

$$\frac{A}{G}(\mathrm{R})$$

Example 2. The *SLD*-system has only one axiom application rule *sld*, defined on every goal G and axiom $\forall(A_1 \wedge \cdots \wedge A_n \rightarrow B)$ such that $G = \theta B$:

$$sld(G, \forall(A_1 \wedge \cdots \wedge A_n \rightarrow B)) = \theta A_1; \ldots; \theta A_n$$

The *SLDNF*-system instead contains the rules (+), (-), (s), and (w). The latter three rules may be applied to negative goals. Since a negative goal may contain many formulae, we decorate (s), (w), (-) by an index j to select L_j in a goal $L_1, \ldots, L_n \vdash_P$. For example:

$$(\text{-}_2)(p(a), p(b) \vdash_P, \forall x.(p(x) \rightarrow q(x) \vee v(x))) \;=\; p(a), q(b) \vdash_P;\; p(a), v(b) \vdash_P$$

[4] From now on, sequences (of goals) will be indicated by $G_1; \ldots; G_n$

The set of proof trees $\mathcal{T}(\mathcal{G}, \mathcal{A}, \mathcal{R})$ is defined inductively as follows:

Definition 24. Every $G \in \mathcal{G}$ is a pt. If $\Pi_1 :: G_1, \ldots, \Pi_n :: G_n$ are pts and $R(G, A) = G_1; \ldots; G_n$, then the following is also a proof tree:

$$\frac{\Pi_1 \quad \Pi_n}{\dfrac{G_1; \ldots G_n; \quad A}{G}(\mathrm{R})}$$

The definitions of *axiom, assumption, proof, consequence* of a proof tree are analogous to the ones given previously in section 2.

Next we restrict to sets $\mathcal{G}$ of goals for which a notion of substitution as an answer/result of the computation makes sense. We assume as well that axioms and rules are not affected by substitutions. Moreover, for the sake of simplicity, we will treat possible eigenvariables as constants.

Remark that, under the previous definitions, Π may be in $\mathcal{T}(\mathcal{G}, \mathcal{A}, \mathcal{R})$, while $\theta\Pi$ is not. To ensure this, we introduce the following

Definition 25. We say that an AAR system $\langle \mathcal{G}, \mathcal{A}, \mathcal{R} \rangle$ is *closed under substitution* if $G \in \mathcal{G}$ entails $\theta G \in \mathcal{G}$ and, for all $R \in \mathcal{R}$ and $A \in \mathcal{A}$, if $R(G, A)$ is defined, then, for every substitution θ, $R(\theta G, A)$ is defined and $R(\theta G, A) = \theta R(G, A)$

Note that the set of values of $R(G, A)$ is connected to the concept of *definiens* and closure under substitution is related to *A-sufficient substitutions* in [11], see section 6 for more details.

Proposition 26. *If $\langle \mathcal{G}, \mathcal{A}, \mathcal{R} \rangle$ is closed under substitution, so is $\mathcal{T}(\mathcal{G}, \mathcal{A}, \mathcal{R})$, i.e. $\Pi \in \mathcal{T}(\mathcal{G}, \mathcal{A}, \mathcal{R})$ entails $\theta\Pi \in \mathcal{T}(\mathcal{G}, \mathcal{A}, \mathcal{R})$*

Therefore the following relations, relying on the notion of substitution, and thus requiring the closure condition, can be defined in the same way as for SLD: $\leq$, $\equiv$, $\preceq$.

Similarity among proof trees is extended to take into account sharing the same *axiom/rule-occurrences*, where an axiom/rule-occurrence is a triple $\langle p, A, R \rangle$ such that p is a path from the root to a node containing an axiom A applied by a rule R.

However, in general $\sim$ and $\leq$ do not satisfy the regularity property 5, so that a similarity class may contain many non-equivalent maximal proof trees and the notion of mgpt becomes problematic.

5.2 Regular *AAR* Systems

Now, let us consider how we could approach the following search problem: given a goal $G \in \mathcal{G}$, find a proof $\Pi :: \theta G$ for some substitution θ. First of all, we need a method to compute one-step continuations (or resolvents, in a sense to be made precise below). Since our aim is to compute substitutions, in a first approximation we may say the following: given $G \in \mathcal{G}$, $A \in \mathcal{A}$ and $R \in \mathcal{R}$, A can

be *applied* to G by R iff there is θ s.t. $R(\theta G, A) = G_1; \ldots; G_n$, in other words we can build the continuation

$$\frac{G_1; \cdots; G_n \; ; \; A}{\begin{array}{c}\theta G \\ \theta \Pi\end{array}} (\mathrm{R})$$

Using the above notion of *application* of an axiom by a rule, we identify applications as one-step continuations. A possibility would be then to try all possible applications, but this would be doomed to incompleteness unless we allow to backtrack on substitutions. This point is clearly exemplified by the lack of regularity (and also of closure under substitution) detected in $SLDNF$-derivations.

Hence we analyse the property of *regularity* and we link it to the notion of *abstract SLD-resolution*; we then show how a SLD-like search strategy turns out to be complete for regular search spaces.

Definition 27. A set $\mathcal{S}$ of proof trees is a *regular search space* iff, for every similar $\Pi_1, \Pi_2 \in \mathcal{S}$, there is a $\Pi \in \mathcal{S}$ such that $\Pi_1 \leq \Pi$ and $\Pi_2 \leq \Pi$.

For regular search spaces, the property 5 of section 2 holds by definition. Moreover, we require that substitutions are well-behaved, so as the equivalent of proposition 6 can be established. Namely, we assume that there is an (non negative integer) measure $m(\Pi)$ such that $\Pi_1 \leq \Pi_2$ entails $m(\Pi_1) > m(\Pi_2)$, provided $\Pi_1 \not\equiv \Pi_2$.

In a regular search space $\mathcal{S}$, the notion of *mgpt* can be defined as for the SLD-system, and the related properties persist. In particular, for any such space $\mathcal{S}$, the following definition is sensible.

Definition 28. Let $\mathcal{S}$ be a regular search space and $T(\mathcal{S}, G)$ be the set of the proof trees $\Pi :: \theta G \in \mathcal{S}$ (where G is a goal); define $Gen(\mathcal{S})$ and $Gen(\mathcal{S}, G)$ to be the corresponding sets of mgpts.

As in Section 3, $\langle Gen(\mathcal{S}, \mathcal{G})/\equiv, \preceq \rangle$ corresponds to search trees in $\mathcal{S}$. Indeed, we introduce an abstract version of a canonical one-step continuation of a pt (see section 2, as the nodes covering it, i.e. its immediate successors. More precisely, we say that a one-step continuation

$$\Pi_1 = \frac{G_1; \cdots; G_n \; ; \; A}{\begin{array}{c}\theta G \\ \theta \overline{\Pi}\end{array}} (\mathrm{R})$$

is *canonical* iff, for every other *similar* continuation

$$\Pi_2 = \frac{H_1; \cdots; H_m \; ; \; A}{\begin{array}{c}\sigma G \\ \sigma \Pi\end{array}} (\mathrm{R})$$

we have that $\Pi_2 \leq \Pi_1$. Thus we have:

$$(*) \quad m = n \text{ and there exists } \delta \text{ s.t. } \sigma = \delta\theta \text{ and } H_i = \delta G_i$$

Regularity ensures that for every goal and applicable rule, there is unique canonical continuation. We can also look at that through an *abstract SLD-resolution method*, denoted by *Res*. Indeed $T(\mathcal{G}, \mathcal{A}, \mathcal{R})$ is regular iff there exists an operator *Res* satisfying the following property:

> $Res(G, A, R) = \langle \theta, G_1; \ldots; G_n \rangle$ iff $R(\theta G, A) = G_1; \ldots; G_n$, and for every other σG s.t. $R(\sigma G, A) = H_1; \ldots; H_m$ the forementioned property $(*)$ holds.

For example, the usual *SLD*-resolution is based on the existence of mgus.

This operator *Res* is an abstraction for $\mathcal{T}(\mathcal{G}, \mathcal{A}, \mathcal{R})$ of the *SLD*-step; given the goal G, new goals are produced in the most general way. Yet, it must be remarked that now backtracking is essentially *bidimensional*, since axiom/rule-occurrences must be taken into account. Eventually the operator imports all the search properties of Prolog, in particular the independence of the selection function. Hence, for every selection function F, the corresponding F-search tree with root G contains the same proofs contained in $Gen(\mathcal{S}, G)$.

Summarizing, an *AAR* system is *regular* iff the set of its proof trees is a regular search space. In a regular *AAR* system, any F-search strategies work as in the *SLD*-system, and the same main results hold.

5.3 Soundness and Completeness Issues for *AAR* Systems

The regularity property guarantees the completeness of an *AAR* system, in the following sense. Let $G \in \mathcal{G}$ be a goal such that there is a proof $\Pi :: \theta G$; then, for every selection function F, in the F-search tree starting from G there is a success node containing a proof Π^* such that $\Pi \leq \Pi^*$. In other words, completeness is relative to the set of the proofs of the system considered.

We propose *AAR* systems as a generalization of logic programs; from this point of view, an *AAR* system is not to be considered as a general logical system, but as a *small* part of some more general (logical) system adequate to solve, in an efficient way, a *particular* class of problems. In this sense, we may look at standard *SLD*-systems as small systems suitable for proving definite goals using Horn axioms. Indeed, the completeness of *SLD*-systems can be read as follows; *SLD*-systems are sufficient to solve all the problems which are solvable in full classical (intuitionistic, minimal) logic, if we restrict to the problems which can be formulated in this language. Uniform proofs [20] are a second example: such a proof procedure is complete for the language of hereditary Harrop formulae w.r.t. intuitionistic derivability.

More precisely, we say that an *AAR* system $\langle \mathcal{G}, \mathcal{A}, \mathcal{R} \rangle$ is *embedded* in a logical system $\mathcal{L}$ if the goals can be interpreted as formulae of the latter and the rules

are sound, i.e. for every axiom $A \in \mathcal{A}$ and every $G \in \mathcal{G}$, $R(G, A) = G_1; \ldots; G_n$ entails $G_1, \ldots, G_n \vdash_{\mathcal{L}} G$. Of course, we assume that the axiom application rules are sound, that is, they are admissible in the logical system.

We will distinguish two kinds of axioms: *logical* axioms, i.e. formulae valid in the logic $\mathcal{L}$ under consideration, and *program* axioms (for example, Horn clauses in logic programs). Only some kinds of formulae are *allowed* as logical or program axioms. The set of the allowed logical axioms may be empty (as in the SLD-system), or contain some particular kind of axioms (such as the ones used in switch or weakening rules in $SLDNF$). In those cases, no logical axioms or only a few of them are used, because part of the logical system $\mathcal{L}$ is implicit in the application rules $\mathcal{R}$.

For every set P of program axioms, and every goal G, a computation means searching for proofs $\Pi :: \theta G$ using the axioms of P and the allowed logical axioms (if any). If the system is regular, then it will compute a proof in $T(\mathcal{G}, \mathcal{A}, \mathcal{R})$, provided there exists one. But this does not imply the completeness w.r.t. the underlying logical system, which can be defined as follows.

Definition 29. An AAR system $(\mathcal{G}, \mathcal{A}, \mathcal{R})$ embedded in a logical system $\mathcal{L}$ is *complete* with respect to $\mathcal{L}$ if, for every set P of program axioms and every goal G, if there is a proof of θG from P in $\mathcal{L}$, then there is a proof $\Pi :: \theta G$ in $T(\mathcal{G}, \mathcal{A}, \mathcal{R})$ with program axioms from P.

Take, as an example, the $SLDNF$-system: are there classes of axioms and goals such that this AAR system is complete with respect to minimal logic (MIN)? A partial answer is negative, in the following sense:

Proposition 30. *There is a normal program P and a goal L such that L is derivable from $Comp(P)$ in minimal logic, but there is no pt in the $SLDNF$ system.*

Proof. Let L be $\neg p$ and P the following program:

$$
\begin{aligned}
&Ax1 \quad p \leftarrow q, r.\\
&Ax2 \quad q \leftarrow a.\\
&Ax3 \quad r \leftarrow \neg a.\\
&Ax4 \quad a \leftarrow a.
\end{aligned}
$$

It is easy to check that, although $\neg p$ has a minimal proof from $Comp(P)$, every pt is infinite. This depends on the absence of the rules for minimal negation.

5.4 Examples of *AARs*

Our work on AARs originated from an analysis of SLD and $SLDNF$-systems; we are just beginning to address the problem of giving some general theory of AAR systems. We conclude this section just with some examples, to illustrate the general idea and some related problems. The first example is motivated by the possibility of proof-theoretically sound proof transformations. In the second example we show how to work with schemas and, in the last one, how to formalize traditional sequent systems.

Fibonacci Numbers in an Extended *SLD*-System. This example is developed in an extended version of the *SLD*-system, where we allow compound formulae. The *SLD*-system has only the axiom application rule *sld*, which leads to a regular system. We generalise such systems to the case where $A_1, \ldots, A_n, B$ are conjunctions of atoms. At first glance, it seems that the explicit introduction of $\wedge$ does not enrich the *SLD*-system, since conjunctions can be treated by commas. Nevertheless, we obtain a richer framework, since it turns out that we have a more flexible way of applying the axioms. Let us consider the usual Horn axioms for computing Fibonacci numbers:

$$\begin{array}{ll} Ax0 & f(0, s(0)) \\ Ax1 & f(s(0), s(0)) \\ Ax2 & \forall x, a, b, c.(f(x, a) \wedge f(s(x), b) \wedge +(a, b, c) \rightarrow f(s(s(x)), c)) \end{array}$$

Using the possibility of compound formulae, we can rewrite the above axioms as follows:

$$\begin{array}{ll} Ax1 & f(0, s(0)) \wedge f(s(0), s(0)) \\ Ax2 & \forall x, a, b, c.(f(x, a) \wedge f(s(x), b) \wedge +(a, b, c) \rightarrow f(s(x), b) \wedge f(s(s(x)), c)) \end{array}$$

An example of a proof tree is the following

$$\dfrac{\dfrac{\dfrac{\text{Ax1}}{f(0,s(0)) \wedge f(s(0),s(0));}\,sld \quad \dfrac{\cdots}{+(s(0),s(0),s(s(0)));}\,sld}{f(s(0),s(0)) \wedge f(s(s(0)),s(s(0)));}\,Ax2\,sld \quad \dfrac{\cdots}{+(s(0),s(s(0)),s(s(s(0))));}}{f(s(s(0)),s(s(0))) \wedge f(s(s(s(0))),s(s(s(0))))}\,Ax2\,sld$$

Many unfold/fold transformations of logic programs can be treated similarly.

Using Schemata. One can easily extend the *SLD*-system to compound formulae, by considering also logical axioms of the following form:

$$\begin{array}{llll} (\wedge) & \forall(A \wedge B \rightarrow A \wedge B) & (\vee_1) & \forall(A \rightarrow A \vee B) \\ (\vee_2) & \forall(B \rightarrow A \vee B) & (\exists) & \forall(A(x) \rightarrow \exists x.A(x)) \end{array}$$

Applying *sld* to $(\wedge)$, $(\vee_1)$, $(\exists)$ ($(\vee_2)$ is omitted, since it is dual to $(\vee_1)$), we obtain the usual introduction rules in a style similar to natural deduction:

$$\dfrac{\theta A;\ \theta B;\ \forall(A \wedge B \rightarrow A \wedge B)}{\theta(A \wedge B)}\,\text{sld} \qquad \dfrac{\theta A;\ \forall(A \rightarrow A \vee B)}{\theta(A \vee B)}\,\text{sld} \qquad \dfrac{\theta A(t);\ \forall(A(x) \rightarrow \exists x.A(x))}{\theta \exists x.A(x)}\,\text{sld}$$

The difference with respect to natural deduction is that every inference is decorated by the applied logical axiom and *substitutions* are present. Notice that

we have an infinite number of logical axioms. Then we would need an infinite backtracking. Nevertheless, for a goal like, e.g. $h(x) \wedge k(g(y), x)$, it is possible to consider *only* the most general continuation related to the axiom $\forall x, y.(h(x) \wedge k(g(y), x) \rightarrow h(x) \wedge k(g(y), x))$; in this way no solution of the goal is lost, even if we do not backtrack on the other applicable axioms.

This means that we can consider $\forall(A \wedge B \rightarrow A \wedge B)$ as a single *schema* and compute the canonical continuations by the first of the following *resolution rules for the schemata* $(\wedge)$, $(\vee_1)$, $(\vee_2)$, $(\exists)$. In the last one n must be a new variable, in order to achieve a most general continuation:

$$\begin{array}{ll}
Res(h \wedge k, (\wedge), sld) = h, k & appl.ax.\ \forall(h \wedge k \rightarrow h \wedge k) \\
Res(h \vee k, (\vee_1), sld) = h & appl.ax.\ \forall(h \rightarrow h \vee k) \\
Res(h \vee k, (\vee_2), sld) = k & appl.ax.\ \forall(k \rightarrow h \vee k) \\
Res(\exists x.h(x), (\exists), sld) = h(n) & appl.ax.\ \forall(h(x) \rightarrow \exists x.h(x))\ (n\ new)
\end{array}$$

For every goal G and every schema S, $Res(G, S, sld)$ computes both the applied axiom (namely the instance of the schema matching the goal) and the corresponding canonical continuation. For example, using Res, the continuation of a proof tree Π with selected goal $h(x) \wedge k(g(y), x)$ is:

$$\frac{h(x);\quad k(g(y), x);\quad \forall x, y.(h(x) \wedge k(g(y), x) \rightarrow h(x) \wedge k(g(y), x))}{\begin{array}{c} h(x) \wedge k(g(y), x) \\ \Pi \end{array}}(sld)$$

Note that the proof tree does not contain the schema, but its first-order instance $\forall x, y.(h(x) \wedge k(g(y), x) \rightarrow h(x) \wedge k(g(y), x))$ computed by Res. Note also that other instances of the schema can be applied to the above goal. Therefore, by means of Res, we generate a subspace of the search space that we would have obtained by backtraking on all the instances. For every goal G, we obtain a regular subspace containing all the answers for G, where similarity is related to the instances (of the schemata) occurring in proof trees.

The situation is more difficult w.r.t. the elimination rules. For example, $\wedge$-elimination can be obtained as an application of a schema as $\forall(A \wedge B \rightarrow A)$. But in this case we cannot obtain a general finite resolution method. Indeed, applying the above to a goal A, we have infinitely many choices of B. Even more serious problems arise while trying to find AARs for the other elimination rules. The fact is that elimination rules may work on assumptions and assumptions are not explicit in the goals. A way to solve this problem exploiting resolution over schemata is to use sequents, distinguishing positive and negative goals and providing more rules, as shown in the next subsection.

Last, it is immediate to see that the usual presentation of first-order systems is not regular (according to our notion of regular AAR system); to recover regularity we need either an AAR or a second-order presentation. For example, let us consider the following instances π_1 and π_2 of the $\wedge$-introduction rule of natural calculus:

$$\frac{p(x) \qquad q(x)}{p(x) \wedge q(x)} \qquad\qquad \frac{u(x) \qquad v(x)}{u(x) \wedge v(x)}$$

π_1 and π_2 apply the same schema, hence they are similar; but there is no *first-order* π such that $\pi_1 \leq \pi$ and $\pi_2 \leq \pi$. Using *AARs*, π_1 and π_2 become decorated by different axioms and are no longer similar. Moreover, for any two similar proof trees, we have a more general form including them.

Simulation of Classical Sequent Calculus with *AARs*. In this section we present a version of the classical (first-order) sequent calculus as *AARs*, using schemata. Here goals have the form $\Gamma \vdash \Delta$, where the sequents are lists of formulae. Contrary to the usual calculus, we search for instances $\theta\Gamma \vdash \theta\Delta$. Schemata and their instances are written in a suitable metalanguage. The following conventions are used: the comma "," induces a splitting of the premises, while ";" creates a single sequent. Formulae annotated with "-" belong to the left of the turnstile, while unannotated ones stay on the right. This corresponds to using the following axiom application rules, where $\pm(\theta B)$ occurs in an integer position $\pm j$ (with the same sign), accordingly in Γ or Δ, and $(\Gamma \vdash \Delta)[\pm j/\Sigma]$ denotes the deletion of the formula at $\pm j$ and the insertion of the formulae of Σ in Γ or Δ, according to their sign:

1) $R_j(\Gamma \vdash \Delta, \forall(A_1, \ldots, A_n \rightarrow \pm B)) = (\Gamma \vdash \Delta)[\pm j/\theta A_1] \ldots (\Gamma \vdash \Delta)[\pm j/\theta A_n]$
2) $R_j(\Gamma \vdash \Delta, \forall(A_1; \ldots; A_n \rightarrow \pm B)) = (\Gamma \vdash \Delta)[\pm j/(\theta A_1, \ldots, \theta A_n)]$

If $\pm B$ is $+\forall x H(x)$ or $-\exists x H(x)$, R_j replaces x by an *eigenvariable* ($\vdash_P$ is annotated by the eigenvariables P, as in (4)). If $\pm B$ is $-\forall x H(x)$ or $+\exists x H(x)$, R_j replaces x by a *new variable*, to obtain a most general continuation. The rules that apply *W-L* and *W-R* (weakening) and *AX* involve pairs of indices, i.e. they are of the form $R_{i,j}$.

Next, we list the schemata below:

$$
\begin{aligned}
&\rightarrow -R : \forall(-H; K \Rightarrow H \rightarrow K)\\
&\rightarrow -L : \forall(H, -K \Rightarrow -(H \rightarrow K))\\
&\wedge -R : \forall(H, K \Rightarrow H \wedge K)\\
&\wedge -L_1 : \forall(-H \Rightarrow -(H \wedge K))\\
&\wedge -L_2 : \forall(-K \Rightarrow -(H \wedge K))\\
&\vee -L : \forall(-H, -K \Rightarrow -(H \vee K))\\
&\vee -R_1 : \forall(H \Rightarrow H \vee K)\\
&\vee -R_2 : \forall(K \Rightarrow H \vee K)\\
&\neg -L : \forall(H \Rightarrow -(\neg H))\\
&\neg -R : \forall(-H \Rightarrow \neg H)\\
&\forall -L : \forall(-H \Rightarrow -\forall x H)\\
&\forall -R : \forall(H \Rightarrow \forall x H)\\
&\exists -L : \forall(-H \Rightarrow -\exists x H)\\
&\exists -R : \forall(H \Rightarrow \exists x H)
\end{aligned}
$$

$$W-L: \forall(-H \Rightarrow -H, -K)$$
$$W-R: \forall(H \Rightarrow H, K)$$
$$C-L: \forall(-H, -H \Rightarrow -H)$$
$$C-R: \forall(H, H \Rightarrow H)$$
$$AX: \quad \forall(\Rightarrow -H, H)$$

A resolution rule for the above schemata, except AX, works as explained in the previous subsection: a resolution step substitutes the metavariables of the 'applied' schema by the formula(s) occurring in the goal, in the position(s) indicated by the rule; in this way, it computes a single *instance* and the corresponding canonical continuation.

For AX, if the rule is $R_{i,j}$, the goal $\Gamma \vdash \Delta$, the formula indicated by $-i$ A and the one indicated by $+j$ B, and σ is a *mgu* of A, B, then the resolvent is $\sigma\Gamma \vdash \sigma\Delta$.

Backtracking is performed mainly on the indexes. Indeed, if we exclude contraction (i.e. C-L and C-R), at most one of the schemata can be applied by a rule R_j (as is well-known, contraction cannot be eliminated, but there are presentations where its use is very much reduced [6]). Moreover, only three schemata can be applied by a rule $R_{i,j}$.

Example 3. We show the quest for an answer substitution for u, v such that the following sequent is provable:

$$r(a) \vee r(b), \forall x.(r(x) \rightarrow h(x)) \vdash h(u), h(v)$$

where a, b are constants. One of the nodes in the search-tree is, for example:

$$\dfrac{\dfrac{\dfrac{r(a) \vdash r(n_1);\ r(a), h(n_1) \vdash h(u), h(v);\ aa3}{r(a), r(n_1) \rightarrow h(n_1) \vdash h(u), h(v);\quad aa2}R_2}{r(a), \forall x.(r(x) \rightarrow h(x)) \vdash h(u), h(v);}R_2 \qquad \dfrac{\dfrac{\dfrac{r(b) \vdash r(n_2);\ r(b), h(n_2) \vdash h(u), h(v);\ aa4}{r(b), r(n_2) \rightarrow h(n_2) \vdash h(u), h(v);\quad aa2}R_2}{r(b), \forall x.(r(x) \rightarrow h(x)) \vdash h(u), h(v);\ aa1}R_2}{r(a) \vee r(b), \forall x.(r(x) \rightarrow h(x)) \vdash h(u), h(v)}R_1$$

where $aa1$ is the instance $-r(a), -r(b) \Rightarrow -(r(a) \vee r(b))$ of the schema $\vee$-L, $aa2$ is the instance $\forall x.(-(r(x) \rightarrow h(x)) \Rightarrow -\forall x.(r(x) \rightarrow h(x)))$ of the schema $\forall$-L, and so on; n_1, n_2 are new variables. From this node, we can apply AX. We can select any one of the open goals, for example $r(a), h(n_1) \vdash h(u), h(v)$. Applying $Res(R_{2,1}, r(a), h(n_1) \vdash h(u), h(v), AX)$, the indicated formulae are $h(n_1)$ and $h(u)$ and we obtain the substitution $\{u/n_1\}$. The new goals of this continuation are $r(a) \vdash r(n_1)$, $r(b) \vdash r(n_2)$ and $r(b), h(n_2) \vdash h(n_1), h(v)$, and now only one successful continuation can be reached (in many ways), giving rise to a proof:

$$\Pi :: r(a) \vee r(b), \forall x.(r(x) \rightarrow h(x)) \vdash h(a), h(b).$$

Applying $Res(R_{2,2}, r(a), h(n_1) \vdash h(u), h(v), AX)$, we get a proof:

$$\Pi :: r(a) \vee r(b), \forall x.(r(x) \rightarrow h(x)) \vdash h(b), h(a).$$

The above is only a first and rough sketch. The study of infinite *AARs* using first-order schemata is part of our future work.

6 Related Work

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

6.1 Resolution

Gallier [9] 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 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 Γ, Δ 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 clauses 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 [26] 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* [22] is a Gentzen system that employs sequents of the form $\Gamma \vdash A$ to perform theorem proving in the non-Horn fragment. Inference 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

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

In a refutational setting U can be taken to be empty, and a set of clauses 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 [7], among other things, stresses the similarities between resolution and tableaux systems, where the usual operations on semantic tableaux à la Smullyan are seen as skolemization steps until resolution 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.

6.2 Logic Programming

The first step is, through simple classical equivalences, to view Horn clauses positively as rules and goals as existentially closed conjunctions of atoms to be proved by those assumptions. Historically this can probably be dated back to Gabbay and Reyle [8]. As mentioned before, we can distinguish two approaches:

1. Clauses as axioms (programs as theories) and some form of Gentzen calculus to infer goals, as very clearly expressed in a series of papers by Miller et al. [20]. The same idea is presented in [24] but w.r.t. natural deduction, where this interpreter is shown to be equivalent to normal derivations in minimal logic. 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; this explains the use of the full (minimal) natural deduction.
2. Clauses as rules [10]: 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 continuous 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 as done above. Differently from us, the authors define a specialized calculus, called *linear derivation*, which proves pairs of the form $\langle G, \theta \rangle$, and demonstrate it to be sound and complete w.r.t. $C(P)$. A linear derivation is essentially a SLD-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 and it is the basis of the sequel of the paper, where [11] a definitional approach to logic programming is sketched: the relationship with our calculus is analyzed in section 6.3.

Regarding negation-as-failure Stärk [27] 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 Σ is a set of equations and Γ of literals

$$\frac{\Sigma \vdash s = t\ ; \quad s = t, \Gamma \vdash \Delta}{\Sigma, \Gamma \vdash \Delta} \qquad \frac{\Sigma \vdash A\ ; \quad A, \Gamma \vdash}{\Sigma, \Gamma \vdash}$$

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

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

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

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 models of the completion. Furthermore a completeness result is proved w.r.t. $SLDNF$-resolution for programs satisfying the cut-property [28].

Harland [14] 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 correspond to our switch rule (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 non-monotonic nature of NF, in the positive fragment weakening is restricted to introduce only already provable formulae on the left. In the negative system structural rules, cut included, are not eliminable.

Note that this is not strictly 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 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 [13] to be sound and complete w.r.t. finite failure and $SLDNF$-resolution (extended to first-order 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.

6.3 Regularity, Anti-Unification, A-Sufficiency

For first-order terms, regularity is analogous to the problem of *generalization* or *anti-unification* [16]. Addressed by Reynolds and Plotkin in 1970, it has become popular in AI under the name of 'explanation-based generalization'. In the lattice of terms under the subsumption ordering, unification [generalization] corresponds to lower [upper] bounds, and in particular mgus [msgs] to glb [lub]. Under the propositions-as-types interpretation, proof trees are terms of a certain type (the goal to be proven). Yet, due to the unpleasant properties of higher-order [anti]unification, the lattice structure can be preserved only by restricting to *higher-order patterns*, long $\beta\eta$-normal forms with constrained occurrences of free variables. Pfenning [23] has given [anti]unification algorithms in the Calculus of Constructions. There regularity and the existence of a mgpt come from anti-unification of higher-order patterns. In our approach similarity is the primitive notion and it may end up either in regular or not regular systems. The above would suggest the question whether every regular AAR system might be described in terms of patterns.

The calculus $D(P)$ [11] is obtained by adding a definitional rule to $C(P)$ (see [3] for a richer set of rules and the programming language $GCLA$ based on them). Let the *definiens* of an atom A be the set $D(A) = \{\theta Y \mid B \Leftarrow Y, A = \theta B\}$. Then, given the proviso of *A-sufficiency*: for all $\theta, D(\theta A) = \theta(D(A))$, we have the rule:

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

Note that although G is derived in $D(P)$ using $(P\vdash)$, the former may not be a logical consequence of the program, nor of any standard extension known in the literature like $comp(P)$ or $CWA(P)$. Indeed the system does not need a logical language at all and has, as a semantics, the theory of *partial inductive definitions* [12].

It is clear that the AAR systems are very strictly related to the definitional approch to logic programming [11], though the latter seems to be more powerful. While we postpone a comparison of the two systems at the more abstract version (generalized AARs versus partial inductive definitions), we may formulate a first result showing the containment of the $SLDNF$-system into $D(P)$, under the proviso of *A-sufficiency*.

Proposition 31. *Let Π be a $SLDNF$-proof of L in $T(Comp(P), L)$, which we assume to be closed under substitution: then $D(P) \vdash L$.*

Proof. A simple induction on Π, using the closure under substitution to ensure the proviso in the application of $(P\vdash)$.

That the other direction does not hold is shown by the counter-example in the proof of (30), which is provable in $D(P)$.

Moreover, there is a strong relationship between regularity and *A-sufficiency*: namely, given a fixed a predicate definion A, in a logic program, every A-sufficient substitution generates a clause in a (not necessarly unique) regular splitting of the failure axiom of A. Thus, if θ is A-sufficient, the rule $\theta A \leftarrow \ldots$ is regular. It seems that the clauses produced this way may not be a total covering of the terms in the language of the program, and that splitting can solve cases where $GCLA$ computations do not find the needed sufficient substitutions.

7 Conclusion and Future Work

From our original goal of giving a proof-theoretic reconstruction of logic programming, as started in [21], we are now in the position of proposing a significant enrichment of the logic programming paradigm. This alone is good evidence of the fruitfulness of the proof-theoretic approach. Moreover, this itinerary has been common to other researchers: by stressing the constructive features of logic programming [20] has formulated the notion of uniform proof. Similarly, [10], [11] have devised a definitional approach to logic programming.

Analogously, from our analysis of SLD and $SLDNF$-resolution, we have elicited the properties (regularity, closure under substitutions et al.) that can turn any AAR system into a Prolog-like programming language.

Concluding, we believe that our proof theoretic interpretation gives a clear explanation of some well-known phenomena and suggests some interesting research directions. We plan to develop our work towards a general theory of regular AARs: many other areas deserve investigation:

- **LF.** We need to understand whether the theory of regular search spaces can serve as a (substantially weaker) logical framework in the sense of LF, and conversely, whether the latter can be shown to be regular.
- **Partial Evaluation.** The positive and negative AAR systems developed to formalize SLD and $SLDNF$-resolution, especially the ordering on proof trees, seems a suitable tool to proof-theoretically reconstruct fold/unfold transformations and partial evaluation in a substantially neater way than in [18].
- **Abduction.** Analogously, the same tools can provide a basis for abduction: the open assumptions of a goal can serve as the abducted premise.
- **Domain Theory.** Remark that the notion of regularity corresponds in order-theoretic terms to a directed set in a partial order: if every directed set has a lub (a mgpt), the partial order is a *pre-domain*. Thus an AAR system is a regular search space provided that its set of pts is a domain. This opens the possibility of providing a domain-theoretic semantics to general AARs.
- **Relations with $GCLA$.** It is possible that the algorithms for A-sufficiency may be useful for finding regular splitting, and vice-versa our term covering approach may provide richer A-substitutions.

- **A Programming Language.** Provide a first experimental implementation of a logic language where the structure of clauses and goals is arbitrary, provided it can be shown to be regular.

Acknowledgment. We would like to thank Roy Dyckhoff and Frank Pfenning for their valuable comments on earlier versions of this paper.

References

1. 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.), pp. 89–148, Morgan Kaufmann, 1988.
2. Apt K.A.: An Introduction to Logic Programming. In: *Handbook of Theoretical Computer Science*, Leuween J. (ed.), Elsevier 1990.
3. Aronsson M., Eriksson L-H., Gäredal A. & Olin P.: The Programming Language GCLA: a Definitional Approach to Logic Programming. *New Generating Computing* **7**, pp. 381–404, 1990.
4. Barbuti R., Mancarella P., Pedreschi D. & Turini F.: A Transformational Approach to Negation in Logic Programming. *Journal of Logic Programming* **8**, pp. 201–228, 1990.
5. Clark K.L.: Negation as Failure. In: *Logic and Data Bases*, Gallaire H. & Minker J. (eds.), Plenum Press, New York, pp. 293–322, 1978.
6. Dyckhoff R.: Contraction-Free Sequent Calculi for Intuitionistic Logic. *Journal of Symbolic Logic* **57**, pp. 795–807, 1992.
7. Fitting M.: *First-Order Logic and Automated Theorem Proving.* Springer-Verlag, 1990.
8. Gabbay D. M. & Reyle U.: N-Prolog: an Extension of Prolog with Hypothetical Implications 1. *Journal of Logic Programming* **1**, pp. 319–355, 1984.
9. Gallier J.: *Logic for Computer Science, Foundations of Automatic Theorem Proving.* Harper & Row, New York, 1986.
10. Hallnäs L. & Schroeder-Heister P.: A Proof-Theoretic Approach to Logic Programming: Clauses as Rules. *Journal of Logic and Computation* **1**, pp. 261–283, 1990.
11. Hallnäs L. & Schroeder-Heister P.: A Proof-Theoretic Approach to Logic Programming: Programs as Definitions. *Journal of Logic and Computation* **1**, pp. 635–660, 1991.
12. Hallnäs L.: Partial Inductive Definitions. *Theoretical Computer Science* **87**, pp. 115–147, 1991.
13. Harland J.: *On Hereditary Harrop Formulae as a Basis for Logic Programming.* PhD Thesis, Edinburgh 1991.
14. Harland J.: Towards a Static Proof System for Negation as Failure. *Citri/TR-92-49*, University of Melbourne, 1992.
15. Kunen K.: Negation in Logic Programming. *Journal of Logic Programming* **4**, pp. 289–308, 1987.
16. Lassez J-L., Maher M.J. & Marriot K.: Unification Revisited.: In: *Foundations of Deductive Databases and Logic Programming.* Minker J. (ed.), Morgan Kaufmann, pp. 587–626 , 1988.
17. Lloyd J.W.: *Foundations of Logic Programming.* Second Extended Edition, Springer-Verlag, Berlin, 1987.

18. Lloyd J.W. & Shepherdson J.: Partial Evaluation in Logic Programming. *Journal of Logic Programming* **11**, pp. 217–242, 1991.
19. Miller D.: A Logical Analysis of Modules in Logic Programming. *Journal of Logic Programming* **6**, pp. 79–108, 1989.
20. Miller D., Nadathur G., Pfenning F., Scedrov A.: Uniform Proofs as a Foundation for Logic Programming. *Annals of Pure and Applied Logic* **51**, pp. 125–157, 1991.
21. Ornaghi M. & Momigliano A.: A Proof-Theoretic Reconstruction of Logic Programming. Abstract. In: *Workshop on Proof and Types*, Pfenning F. (ed.) JICSLP92, Washington, pp 22–23, 1992.
22. Plaisted D.A.: Non-Horn Logic Programming without Contrapositives. *JJournal of Automated Reasoning* **4**, pp. 287–325, 1988.
23. Pfenning F.: Unification and Anti-Unification in the Calculus of Constructions. *Proc. of LICS91*, pp. 74–85, 1991.
24. Pfenning F.: Dependent Types in Logic Programming. *Types in Logic Programming*, Pfenning F. (ed.), MIT Press, Cambridge, pp. 285–312, 1992.
25. Shepherdson J.C.: Negation in Logic Programming. In: *Foundations of Deductive Databases and Logic Programming*, Minker J. (ed.), Morgan Kaufmann, pp. 19–88, 1988.
26. Snyder W. & Lynch C.: Goal-Oriented Strategies for Paramodulation, *RTA-91*, Book R.V. (ed.), Lecture Notes in Computer Science **488**, Springer-Verlag, pp. 150–161, 1991.
27. Stärk R.: *The Proof-Theory of Logic Programs with Negation.* PhD Thesis, University of Bern, 1992.
28. Stärk R.: Cut-Property and Negation as Failure. *Technical Report*, University of Bern, 1992.