Proof-Theoretic Foundations of Indexing in Logic Programming[*]

Iliano Cervesato, Carnegie Mellon University
iliano@cmu.edu

ABSTRACT

Indexing is generally viewed as an implementation artifact, indispensable to speed up the execution of logic programs and theorem provers, but with little intrinsically logical about it. We show that indexing can be given a justification in proof theory on the basis of focusing and linearity. We demonstrate this approach on predicate indexing for Horn clauses and several formulations of hereditary Harrop formulas. We also show how to refine this approach to discriminate on function symbols as well.

Categories and Subject Descriptors

D.3.1 [**Programming Languages**]: Formal Definitions and Theory—*semantics*; F.3.2 [**Logics and Meanings of Programs**]: Semantics of Programming Languages

General Terms

Logic programming

Keywords

Logic programming, proof-theory, foundations, indexing

1. INTRODUCTION

Logic programming, theorem proving and other incarnations of computational logic straddle two worlds. On the one hand, there is the world of logic, whose universal language and simple rules allow us to give an abstract specification to a problem and to reason about it in a simple and natural way. On the other hand, there is the computational world, centered around the urgency of returning answers fast. Tools like logic programming compilers and automated theorem provers bring these worlds together: they do implement the rules of logic to search for the proofs that justify those answers, but as they do so they overlay onto them a thick layer of heuristics, because naively trying all rules that apply would be hopelessly inefficient. These heuristics yield significant speedups but their relation to logic can be remote: they are hacks of convenience, tolerated by purists solely out of necessity.

In recent years, many of these heuristics have been given a logical justification in terms of proof theory, thereby bringing them back into the fold of logic. Examples include the very idea of goal-oriented proof search, espoused by Prolog for example, which is now understood as an instance of focusing [7, 5] (an earlier explanation, uniform provability [9], is a special case of focusing). Another example is WAM-style compilation of logic programs [2, 3], which is founded on duality and currying. Standard implementation techniques such as left-to-right goal search and first-to-last clause selection have found their foundation in ordered logic [12], while unification combines logic with equality and contextual reasoning [1].

One heuristic that has received little foundational attention is indexing. When goal-oriented proof search reaches an atomic goal, indexing narrows the selection of the program formulas to continue the search with to just those that are sufficiently similar to the goal. One form of indexing that is universally used in logic programming is to consider just those clauses whose head has the same predicate symbol as the atomic goal we are trying to prove. Prolog improves on this by considering the leading function symbol in the first argument of the clause head as well. Indexing is also used in forward logic programming (to quickly identify candidate clauses to fire when a new fact enters the state) and theorem proving (to quickly retrieve relevant lemmas) [10, 13]. So far, indexing has been just one of those things one does to speed up search, without much thought given to its logical status.

In this paper, we show that some forms of indexing can be given a logical pedigree. We do so in the context of backward logic programming, although we believe that our techniques are broadly applicable. The key idea is to pair up each atomic goal and each clause with a proposition, the index, in such a way to force the use of relevant clauses when processing an atomic goal. We rely on polarization and focusing to force derivations: the index is a positive atom, which a focused proof-search discipline resolves immediately. The index has furthermore to be linear to avoid accumulating stale indices.

[*]This paper was made possible by an NPRP grant (NPRP 4-341-1-059, Usable automated data inference for end-users) from the Qatar National Research Fund (a member of the Qatar Foundation). The statements made herein are solely the responsibility of the authors.

$$\frac{}{\Gamma \longrightarrow_\Sigma \boxed{\top}}\ \top_\mathsf{R} \qquad \frac{\Gamma \longrightarrow_\Sigma \boxed{G_1} \quad \Gamma \longrightarrow_\Sigma \boxed{G_2}}{\Gamma \longrightarrow_\Sigma \boxed{G_1 \wedge G_2}}\ \wedge_\mathsf{R}$$

$$\frac{\Gamma, D, \boxed{D} \longrightarrow_\Sigma a^{-}}{\Gamma, D \longrightarrow_\Sigma \boxed{a^{-}}}\ \mathsf{atm_R}$$

$$\frac{\Gamma \longrightarrow_\Sigma \boxed{G}}{\Gamma, \boxed{G \supset a^{-}} \longrightarrow_\Sigma a^{-}}\ \supset_\mathsf{L} \qquad \frac{\vdash_\Sigma t \quad \Gamma, \boxed{[t/x]D} \longrightarrow_\Sigma a^{-}}{\Gamma, \boxed{\forall x.\, D} \longrightarrow_\Sigma a^{-}}\ \forall_\mathsf{L}$$

Figure 1: Focused Provability for Horn Clauses

This technique works well for predicate indexing on Horn clauses and minimalistic hereditary Harrop formulas (a small alteration is needed for more general presentations). It also works well when indexing on function symbols for first-order terms (in any position and at any depth).

The main contribution of this work is to promote indexing to a logic-based, provably correct, program transformation status. This enables us to reason about it with the tools and techniques of logic and to include it in the arsenal of logic-based optimizations and refinements. We do not anticipate this work to play a direct role in implemented systems — the procedural approaches in use deliver excellent performance — but it should help us frame the notion of indexing declaratively.

The rest of the paper is organized as follows: we introduce our approach in Section 2 on the problem of indexing predicate symbols for Horn clauses. We then refine it in Section 3 to function symbols. We expand its scope in Section 4 by applying it to languages based on hereditary Harrop formulas. We discuss relevant work in the literature in Section 5 and outline directions of future work in Section 6.

2. INDEXING PREDICATE SYMBOLS

The language of Horn clauses, which underlies the core of Prolog, is given by the following grammar:

$$\begin{array}{rrcl} \textit{Atoms:} & a^{-} & ::= & p^{-}(\vec{t}) \\ \textit{Goals:} & G & ::= & a^{-} \mid \top \mid G_1 \wedge G_2 \\ \textit{Clauses:} & D & ::= & G \supset a^{-} \mid \forall x.\, D \\ \textit{Programs:} & \Gamma & ::= & \cdot \mid \Gamma, D \end{array}$$

A goal is a conjunction of atoms, a clause is the universal closure of an implication with a goal (the clause's body) as its antecedent and an atom (the clause's head) as its consequent, and a program is a collection of clauses. In the focusing interpretation of backward chaining proof search, all atoms are assigned negative polarity, which we highlight by writing them as a^{-} and similarly adorning predicate symbols. We write $\vec{t}$ for a tuple of terms. The exact term language is unimportant for the present discussion, as we will not examine term-based indexing until Section 3.

Backward chaining derivability for Horn clauses is abstractly expressed by the following two judgments [9],

$$\begin{array}{ll} \Gamma \longrightarrow_\Sigma \boxed{G} & \textit{Goal } G \textit{ is derivable from } \Gamma \textit{ and } \Sigma \\ \Gamma, \boxed{D} \longrightarrow_\Sigma a^{-} & \textit{Clause } D \textit{ derives } a^{-} \textit{ using } \Gamma \textit{ and } \Sigma \end{array}$$

whose defining rules are given in Figure 1. Once the goal has been reduced to an atomic formula a^{-}, rule $\mathsf{atm_R}$ selects a clause D from the program. Its layer of universal quantifiers are instantiated using rule $\forall_\mathsf{L}$, exposing the implication that is processed by rule $\supset_\mathsf{L}$. Notice that it is only at this point that we check that the head of the clause D picked by $\mathsf{atm_R}$ has anything to do with the atomic goal a^{-}. If we think of an atomic goal as a key and of a clause as a lock, we have to walk down a long corridor to check if the key opens the lock. This is fine for an abstract system, like the one in Figure 1, that describes what a valid derivation looks like rather than how to find one. This is however wasteful in an operational refinement that internalizes clause selection and resolves incorrect choices through backtracking: we may end up walking down many long corridors finding each time that the lock is not the right one. Operationally, we want to limit the selection to clauses that will have some chance to solve a^{-}, which excludes any clause whose head has a predicate different from that of a^{-}. But rule $\mathsf{atm_R}$ is agnostic to this. This same agnosticism is carried over in the standard refinement of this system that internalizes backtracking for Horn clauses. We would like the following variant of rule $\mathsf{atm_R}$:

$$\frac{\Gamma, D_p, \boxed{D_p} \longrightarrow_\Sigma p^{-}(\vec{t})}{\Gamma, D_p \longrightarrow_\Sigma \boxed{p^{-}(\vec{t})}}\ \mathsf{atm'_R}$$

where the clause D_p explicitly mentions the predicate in its head. But D_p has no logical meaning. Here, p is an *index*: it allows us to restrict the search to just those clauses that match the head. This is akin to having the lock, or at least a picture of it, at the entrance of each corridor. Indexing is typically implemented by storing the program in a hash table with the index as the hash key. This yields logarithmic (and often constant) look-up cost, a significant improvement over a naive linear scan.

We will give a logical foundation to indexing by turning the problem of selecting matching clauses into a proof search problem of its own, but one that can be computed efficiently. To this end, we associate a new nullary predicate symbol, say i_p for now, to each predicate symbol p^{-} we want to index (recall that we are only indexing predicate symbols in this section). We use i_p to guard each clause with p^{-} as its head, so that, informally, it will assume the form $i_p \supset D$. To unlock the use of D during proof search, we must supply an instance of i_p. We do so by promoting each atomic goal of the form $p^{-}(\vec{t})$ to the embedded implication $i_p \supset p^{-}(\vec{t})$.

This basic idea needs to be refined in two ways. First, we need to guarantee that the proof search for the subgoal i_p in $i_p \supset D$ immediately matches the trigger i_p deposited in the context by an atomic goal $i_p \supset p^{-}(\vec{t})$. We do so by making i_p a *positive* atom, which we will write as p^{+} from now on. This also guarantees a separation of name spaces between the atoms from our original program and the atoms we use as indices. Second, we want to make sure that the instance of the trigger p^{+} added to the context by atomic goal $p^{+} \supset p^{-}(\vec{t})$ does not linger — otherwise, a stale q^{+} could unlock a clause $q^{+} \supset D'$ even though the current goal has the form $p^{-}(\vec{t})$. We avoid this by making an index p^{+} into a *linear* atom, using the linear implication $\multimap$ to manipulate it. Therefore, we will provide a logical foundation to indexing by translating a source system, here the language of Horn clauses, into a fragment of focused linear logic with both

$$\frac{}{\underline{\Gamma};\cdot \longrightarrow_\Sigma \boxed{\mathbf{1}}}\,\mathsf{1_R} \qquad \frac{\underline{\Gamma};\cdot\longrightarrow_\Sigma \boxed{\underline{G_1}} \quad \underline{\Gamma};\cdot\longrightarrow_\Sigma\boxed{\underline{G_2}}}{\underline{\Gamma};\cdot\longrightarrow_\Sigma\boxed{\underline{G_1}\otimes\underline{G_2}}}\,\otimes_\mathsf{R}$$

$$\frac{\underline{\Gamma}, p^+ \multimap \underline{D}; q^+, \boxed{p^+\multimap\underline{D}} \longrightarrow_\Sigma a^-}{\underline{\Gamma}, p^+\multimap \underline{D}; q^+ \longrightarrow_\Sigma \boxed{a^-}}\,\mathsf{atm_R}$$

$$\frac{\underline\Gamma;\cdot\longrightarrow_\Sigma\boxed{\underline G}}{\underline\Gamma;\cdot,\boxed{\underline G\supset a^-}\longrightarrow_\Sigma a^-}\,\supset_\mathsf{L} \qquad \frac{\vdash_\Sigma t \quad \underline\Gamma;\cdot,\boxed{[t/x]\underline D}\longrightarrow_\Sigma a^-}{\underline\Gamma;\cdot,\boxed{\forall x.\,\underline D}\longrightarrow_\Sigma a^-}\,\forall_\mathsf{L}$$

$$\frac{}{\underline\Gamma;p^+\longrightarrow_\Sigma\boxed{p^+}}\,\mathsf{init_R} \qquad \frac{\underline\Gamma;p^+\longrightarrow_\Sigma\boxed{a^-}}{\underline\Gamma;\cdot\longrightarrow_\Sigma\boxed{p^+\multimap a^-}}\,\multimap_\mathsf{R}$$

$$\frac{\underline\Gamma;q^+\longrightarrow_\Sigma\boxed{p^+}\quad\underline\Gamma;\boxed{\underline D}\longrightarrow_\Sigma a^-}{\underline\Gamma;q^+,\boxed{p^+\multimap\underline D}\longrightarrow_\Sigma a^-}\,\multimap_\mathsf{L}$$

Figure 2: Indexed Horn Clause Derivability

positive and negative atoms.

Formally, our target language is given by the following grammar:

$$\begin{array}{rrcl}
\textit{Head formulas:} & H & ::= & p^+ \multimap a^- \\
\textit{Goal formulas:} & \underline{G} & ::= & H \mid \mathbf{1} \mid \underline{G_1} \otimes \underline{G_2} \\
\textit{Program formulas:} & \underline{D} & ::= & \underline{G} \supset a^- \mid \forall x.\,\underline{D} \\
\textit{Programs:} & \underline{\Gamma} & ::= & \cdot \mid \underline{\Gamma}, p^+ \multimap \underline{D} \\
\textit{Active indices:} & \Delta & ::= & \cdot \mid p^+
\end{array}$$

Negative atomic formulas stay unchanged. For clarity, we write the name of corresponding syntactic classes as in our source language, but underline them, for example writing $\underline{G}$ for the formula a goal G is mapped to. For emphasis, we highlight indices and the linear implication that introduces them in blue. As commonly done, we understand $\underline{G} \supset a^-$ as $!\underline{G} \multimap a^-$, but take it as primitive.

Derivability in our target system is captured by the following two judgments,

$$\begin{array}{ll}
\underline{\Gamma};\Delta \longrightarrow_\Sigma \boxed{\underline{G}} & \textit{Goal } \underline{G} \textit{ is derivable from } \underline{\Gamma},\ \Delta \textit{ and } \Sigma \\
\underline{\Gamma};\Delta, \boxed{\underline{D}} \longrightarrow_\Sigma a^- & \textit{Clause } \underline{D} \textit{ derives } a^- \textit{ using } \underline{\Gamma},\ \Delta \textit{ and } \Sigma
\end{array}$$

which correspond directly to the judgments of our original system. In this section, we will maintain the invariant that the linear context Δ is either empty or contains exactly one positive atom (an index). A specialization of the standard focusing system of linear logic [5] is displayed in Figure 2. A red border denotes the focus on an asynchronous formula, while the green border realizes an inversion strategy on synchronous formulas — the distinction can however be glossed over unless examining the details of the proofs. The top part of Figure 2 adapts the rules in Figure 1. The bottom part adds rules for handling the embedded implications in the goal (rule $\multimap_\mathsf{R}$) and in the program (rule $\multimap_\mathsf{L}$), and the processing of positive atoms (rule $\mathsf{init_R}$). Notice that positive atoms succeed immediately without triggering any search.

Before we spell out the details of our encoding, consider the following example, which we will use throughout the paper. This is the standard definition of the `append` predicate, written as follows in Prolog.

```
append([], L, L).
append([X|L1], L2, [X|L3]) :- append(L1, L2, L3).
```

It assumes the following form in our source language, where we abbreviate `append` as `app`, and use `nil` and `c` for the empty list and the list constructor respectively.

$$\begin{array}{r}
\forall l. \qquad\qquad \top \supset \mathsf{app}^-(\mathsf{nil}, l, l) \\
\forall x, l_1, l_2, l_3.\, \mathsf{app}^-(l_1, l_2, l_3) \supset \mathsf{app}^-(\mathsf{c}(x, l_1), l_2, \mathsf{c}(x, l_3))
\end{array}$$

To make indexing explicit, we transform programs and goals in our source language into similar entities in our target language. We do so by means of the three-part transformation shown next, where $\ulcorner G \urcorner$ and $\ulcorner \Gamma \urcorner$ denote the encoding of the goal G and program Γ, respectively. The latter relies on $\ulcorner\!\ulcorner D \urcorner\!\urcorner$ which encodes a program formula D. As usual, we highlight indices p^+ in blue.

$$\begin{array}{rcl}
\ulcorner p^-(\vec{t}) \urcorner & = & p^+ \multimap p^-(\vec{t}) \\
\ulcorner \top \urcorner & = & \mathbf{1} \\
\ulcorner G_1 \wedge G_2 \urcorner & = & \ulcorner G_1 \urcorner \otimes \ulcorner G_2 \urcorner \\
\ulcorner\!\ulcorner G \supset p^-(\vec{t}) \urcorner\!\urcorner & = & \ulcorner G \urcorner \supset p^-(\vec{t}) \\
\ulcorner\!\ulcorner \forall x.\, D \urcorner\!\urcorner & = & \forall x.\, \ulcorner\!\ulcorner D \urcorner\!\urcorner \\
\ulcorner . \urcorner & = & \cdot \\
\ulcorner \Gamma, D \urcorner & = & \ulcorner \Gamma \urcorner, H(D) \multimap \ulcorner\!\ulcorner D \urcorner\!\urcorner
\end{array}$$

where $H(D)$ denotes the index of the head of clause D, formally defined as

$$\begin{cases} H(G \supset p^-(\vec{t})) & = \ p^+ \\ H(\forall x.\, A) & = \ H(A) \end{cases}$$

We mnemonically wrote p^+ for the index corresponding to predicate p^-, but there is no reason for these two symbols to be syntactically related. The only requirement is that there is a one-to-one correspondence between indices and the atomic predicates they are intended to index.

Here is the translation of our example:

$$\begin{array}{ll}
\mathsf{app}^+ \multimap & \forall l. \qquad\qquad\qquad \mathbf{1} \supset \mathsf{app}^-(\mathsf{nil}, l, l) \\
\mathsf{app}^+ \multimap & \forall x, l_1, l_2, l_3.\, (\mathsf{app}^+ \multimap \mathsf{app}^-(l_1, l_2, l_3)) \supset \\
 & \qquad\qquad\qquad \mathsf{app}^-(\mathsf{c}(x, l_1), l_2, \mathsf{c}(x, l_3))
\end{array}$$

The goal $\mathsf{app}^-(\mathsf{c}(m, \mathsf{nil}), \mathsf{c}(n, \mathsf{c}(o, \mathsf{nil})), \mathsf{c}(m, \mathsf{c}(n, \mathsf{c}(o, \mathsf{nil}))))$, i.e., `?- append([m],[n,o],[m,n,o])` in Prolog, is translated to

$$\mathsf{app}^+ \multimap \mathsf{app}^-(\mathsf{c}(m, \mathsf{nil}), \mathsf{c}(n, \mathsf{c}(o, \mathsf{nil})), \mathsf{c}(m, \mathsf{c}(n, \mathsf{c}(o, \mathsf{nil}))))$$

We will now show that this transformation maps provable sequents to provable sequents. We start by showing that it is complete, i.e., that whenever a goal is derivable from a program, the same holds for their translation. Note the form of the atomic goal in the second part of this statement.

Lemma 1 (Completeness).

- *If* $\Gamma \longrightarrow_\Sigma \boxed{G}$, *then* $\ulcorner \Gamma \urcorner; \cdot \longrightarrow_\Sigma \boxed{\ulcorner G \urcorner}$.
- *If* $\Gamma, \boxed{D} \longrightarrow_\Sigma a^-$, *then* $\ulcorner \Gamma \urcorner; \cdot, \boxed{\ulcorner\!\ulcorner D \urcorner\!\urcorner} \longrightarrow_\Sigma a^-$.

Proof. The proof proceeds by mutual induction on the derivation $\mathcal{D}$ of the judgment in the antecedent of each statement. The cases for goal derivations (the first statement) ending in rules $\top_{\mathsf{R}}$ and $\wedge_{\mathsf{R}}$ follow immediately from the induction hypothesis. We expand the case where the last rule used in $\mathcal{D}$ is $\mathsf{atm}_{\mathsf{R}}$.

By inversion on this rule, the sequent $\Gamma', D, \boxed{D} \longrightarrow_{\Sigma} a^-$ is derivable where $\Gamma = \Gamma', D$. For such a derivation to exist, it must be the case that $a^- = p^-(\vec{t})$ and that p^- is the head predicate of D, so that $H(D) = p^+$. We can then apply the second induction hypothesis, obtaining that the sequent $\ulcorner \Gamma', D \urcorner; \cdot, \boxed{\ulcorner D \urcorner} \longrightarrow_{\Sigma} a^-$ is derivable. Now, by rule $\mathsf{init}_{\mathsf{R}}$, the sequent $\ulcorner \Gamma', D \urcorner; p^+ \longrightarrow_{\Sigma} \boxed{p^+}$ is also derivable. Combining these two derivations using rule $\multimap_{\mathsf{L}}$ we obtain a derivation of $\ulcorner \Gamma', D \urcorner; p^+, \boxed{p^+ \multimap \ulcorner D \urcorner} \longrightarrow_{\Sigma} a^-$. Recalling that $p^+ = H(D)$, we can now apply rule $\mathsf{atm}_{\mathsf{R}}$ to obtain a derivation of $\ulcorner \Gamma', D \urcorner; p^+ \longrightarrow_{\Sigma} \boxed{a^-}$. Finally, by rule $\multimap_{\mathsf{R}}$, we get a derivation of $\ulcorner \Gamma', D \urcorner; \cdot \longrightarrow_{\Sigma} \boxed{p^+ \multimap a^-}$ which is what we wanted since $\ulcorner a^- \urcorner = p^+ \multimap a^-$.

The cases dealing with program formulas (the second statement) follow by a straightforward use of the induction hypothesis. □

The reverse property, soundness, ensures that, whenever the translation of a sequent is provable, the original sequent is provable as well. It is formalized in the following lemma.

Lemma 2 (Soundness).

- *If* $\ulcorner \Gamma \urcorner; \cdot \longrightarrow_{\Sigma} \boxed{\ulcorner G \urcorner}$, *then* $\Gamma \longrightarrow_{\Sigma} \boxed{G}$.
- *If* $\ulcorner \Gamma \urcorner; \cdot, \boxed{\ulcorner D \urcorner} \longrightarrow_{\Sigma} a^-$, *then* $\Gamma, \boxed{D} \longrightarrow_{\Sigma} a^-$.

Proof. This proof is essentially the inverse of the proof of completeness. The cases for non-atomic goals and program formulas follow by an application of the induction hypothesis. The case for atomic goals is interesting in that it makes an essential use of the relationship between predicate symbols and their index. It also witnesses the strict constraints put in place by focusing.

Assume then that the goal is atomic, so that G has the form $a^- = p^-(\vec{t})$, and therefore $\ulcorner G \urcorner = p^+ \multimap p^-(\vec{t})$. Our assumption is therefore that the sequent $\ulcorner \Gamma \urcorner; \cdot \longrightarrow_{\Sigma} \boxed{p^+ \multimap a^-}$ is derivable. The only rule that can have produced it is $\multimap_{\mathsf{R}}$, from which we deduce that the sequent $\ulcorner \Gamma \urcorner; p^+ \longrightarrow_{\Sigma} \boxed{a^-}$ must also be derivable. In turn, this sequent can only have resulted from an application of rule $\mathsf{atm}_{\mathsf{R}}$. This rule pulls a formula of the form $H(D) \multimap \ulcorner D \urcorner$ from the program. Note that, a priori, $H(D)$ and p^+ can be the same atom or may be different.

Inversion on rule $\mathsf{atm}_{\mathsf{R}}$ yields a derivation for the sequent $\ulcorner \Gamma \urcorner; p^+, \boxed{H(D) \multimap \ulcorner D \urcorner} \longrightarrow_{\Sigma} a^-$ which can only be the product of rule $\multimap_{\mathsf{L}}$. The left premise of this rule has the form $\ulcorner \Gamma \urcorner; H(D) \longrightarrow_{\Sigma} \boxed{p^+}$, which is derivable (using rule $\mathsf{init}_{\mathsf{R}}$) only if $H(D) = p^+$. On the other hand, the right premise has the form $\ulcorner \Gamma \urcorner; \boxed{\ulcorner D \urcorner} \longrightarrow_{\Sigma} a^-$. We can apply the second induction hypothesis which yields that $\Gamma, \boxed{D} \longrightarrow_{\Sigma} a^-$. Now, a simple application of rule $\mathsf{atm}_{\mathsf{R}}$ yields a derivation of $\Gamma \longrightarrow_{\Sigma} \boxed{a^-}$, which is what we were after. □

Observe that the cases we have expanded *force* the selected clause D to have the same index as the goal a^- we are trying to prove. Indexing at work! It is instructive to see what this proof does as a derivation snippet. Using the notation of the target language, it assumes the following form:

$$\dfrac{\dfrac{\dfrac{\dfrac{}{\underline{\Gamma}; p^+ \longrightarrow_{\Sigma} \boxed{p^+}}\,\mathsf{init}_{\mathsf{R}} \qquad \underline{\Gamma}; \cdot, \boxed{\underline{D}} \longrightarrow_{\Sigma} p^-(\vec{t})}{\underline{\Gamma}; p^+, \boxed{p^+ \multimap \underline{D}} \longrightarrow_{\Sigma} p^-(\vec{t})}\,\multimap_{\mathsf{L}}}{\underline{\Gamma}; p^+ \longrightarrow_{\Sigma} \boxed{p^-(\vec{t})}}\,\mathsf{atm}_{\mathsf{R}}}{\underbrace{\underline{\Gamma'}, p^+ \multimap \underline{D}}_{\underline{\Gamma}}; \cdot \longrightarrow_{\Sigma} \boxed{p^+ \multimap p^-(\vec{t})}}\,\multimap_{\mathsf{R}}$$

Every step except for the clause selection in rule $\mathsf{atm}_{\mathsf{R}}$ is forced by the focusing discipline. The choice of the clause $p^+ \multimap \underline{D}$ in rule $\mathsf{atm}_{\mathsf{R}}$ is partially forced: only clauses whose index is p^+ can close the derivation, two steps later using rule $\mathsf{init}_{\mathsf{R}}$.

This derivation snippet can be collapsed into a synthetic rule that, once brought back to the language of Horn clauses, correspond closely to the mythical rule $\mathsf{atm}'_{\mathsf{R}}$ at the beginning of this section. The indexed clause D_p is an abbreviation of $p^+ \multimap D$ and the goal $p^-(\vec{t})$ abbreviates the indexed goal $p^+ \multimap p^-(\vec{t})$. The index p^+ is the hashing key that enable constant clause look-up.

3. INDEXING FUNCTION SYMBOLS

We will now extend the technique just discussed to take into consideration the structure of the terms in atomic formulas. We will limit the discussion to traditional first-order terms, defined by the following grammar:

$$\begin{array}{rrcl} \textit{Terms:} & t & ::= & x \mid f(\vec{t}) \\ \textit{Term tuples:} & \vec{t} & ::= & \cdot \mid \vec{t}, t \end{array}$$

Here, x is a variable, and we write $\mathrm{FV}(t)$ for the free variables in term t. We extend this notation to term tuples and formulas of any kind. In this grammar, f is a function symbol. As usual, a constant is a function symbol applied to zero arguments — we omit the parentheses and the empty term tuple in this case.

Term-based indexing takes into account not just the predicate symbol in an atomic goal and in the head of a clause, but also information about the terms in their arguments. For example, Prolog implementations index the leading function symbol in the first argument of a predicate (this is nil in $\mathsf{app}^-(\mathsf{nil}, l, l)$ and c in $\mathsf{app}^-(\mathsf{c}(x, l_1), l_2, \mathsf{c}(x, l_3))$. Our treatment will be more liberal: we will allow indexing on any fixed argument position of each predicate symbol p^-. We call it the *indexing position* of p^- — it may differ across predicate symbols. For simplicity, we will only consider indexing on the leading function symbol, although this is not a limitation. On the other hand, we shall assume that the set of possible leading function symbols in the indexing position of each predicate is known. Some languages fix these symbols through typing and, even in untyped languages like Prolog, programmers typically expect specific terms in each position. The identification of these symbols is closely related to the coverage problem [14].

We associate an *indexing constant* with each function symbol of interest. For simplicity, we will reuse the function symbol itself as this constant. So, in the example of app^- above, we associate nil to nil and c to $\mathsf{c}(\_,\_)$. We write $\mathrm{IC}(p^-)$ for the set of the indexing constants associated with the leading function symbols in the index position of predicate p^- in a program. For example, when indexing on the first argument of app^-, we have that $\mathrm{IC}(\mathsf{app}^-) = \{\mathsf{nil}, \mathsf{c}\}$ because the definition of app^- forces the leading function symbol in its first argument to be either nil or $\mathsf{c}(\_,\_)$. We will not be concerned with how $\mathrm{IC}(p^-)$ is determined: we assume it is given to us by the programmer or through static analysis. We will use $\mathrm{IC}(p^-)$ in the indexing translation of a Horn clause sequent.

We introduce our approach by examining the three possible indexing positions in the predicate $\mathsf{app}^-(l_1, l_2, l_3)$.

- Let us start with indexing on the first argument of app^-, Prolog style. Therefore, whenever we encounter an atom of the form $\mathsf{app}^-(l_1, l_2, l_3)$, we shall take action on the form of l_1. From the above discussion, $\mathrm{IC}(\mathsf{app}^-) = \{\mathsf{nil}, \mathsf{c}\}$ in this case.

  We first translate the clauses defining app^-, for the moment ignoring subgoals. The idea is to equip the positive atom app^+ associated to predicate app^- with an argument, which will correspond to the leading function symbol (if any) of the atomic goal for this predicate. Because each clause discriminates on one of the two members of $\mathrm{IC}(\mathsf{app}^-)$, each will be guarded by the corresponding instance of app^+:

$$\begin{array}{lll} \mathsf{app}^+(\mathsf{nil}) \multimap & \forall l. & \mathbf{1} \supset \mathsf{app}^-(\mathsf{nil}, l, l) \\ \mathsf{app}^+(\mathsf{c}) \multimap & \forall x, l_1, l_2, l_3. \ulcorner \mathsf{app}^-(l_1, l_2, l_3) \urcorner \supset & \\ & & \mathsf{app}^-(\mathsf{c}(x, l_1), l_2, \mathsf{c}(x, l_3)) \end{array}$$

  Notice that we did not translate (yet) the subgoal $\mathsf{app}^-(l_1, l_2, l_3)$ of the second clause. Because l_1 is a variable, it could be instantiated to either nil or $\mathsf{c}(\_,\_)$: we do not have this information at this stage. We must therefore be prepared to place in the context both the trigger $\mathsf{app}^+(\mathsf{nil})$ if l_1 turns out to be nil or the trigger $\mathsf{app}^+(\mathsf{c})$ if it has the form $\mathsf{c}(\_,\_)$. One way to do so is by translating such unconstrained atomic goal as follows:

$$\begin{array}{lll} (& l_1 = \mathsf{nil} & \otimes (\mathsf{app}^+(\mathsf{nil}) \multimap \mathsf{app}^-(l_1, l_2, l_3)) \\ \oplus & (\exists y, z.\ l_1 = \mathsf{c}(y, z) & \otimes (\mathsf{app}^+(\mathsf{c}) \multimap \mathsf{app}^-(l_1, l_2, l_3))) \end{array}$$

  with the variables y and z abstracting the subterms of l_1 below c. If l_1 is nil, (only) the first disjunct applies and the trigger $\mathsf{app}^+(\mathsf{nil})$ is inserted in the context. If l_1 has the form $\mathsf{c}(t_1, t_2)$, then the second disjunct applies (exclusively), which leads to inserting $\mathsf{app}^+(\mathsf{c})$ in the context. If l_1 is a variable, then each disjuncts will be tried in turn.

- Next, let's index app^- on its second argument, l_2. The clauses for app^- do not put any constraint on this position: in each of them, it appears as a variable that can be instantiated to any term. This implies that, in this case, $\mathrm{IC}(\mathsf{app}^-) = \{\}$: there is nothing to index in l_2. As a consequence, our translation will be the encoding in Section 2: the best we can do is index on the predicate symbol.

- Our last option is to index app^- on its third argument, l_3. The first clause for app^- does not put constraints on this position (it appears as a variable), while the head of the second clause requires that its leading function symbol be c. We handle this situation by guarding all the clauses for app^-, but letting the trigger for the first clause be a variable. We obtain the following encoding, where the translation of the inner goal of the second clause is as earlier, but this time discriminating on l_3 rather than l_2.

$$\begin{array}{lll} \forall i.\ \mathsf{app}^+(i) \multimap & \forall l. & \mathbf{1} \supset \mathsf{app}^-(\mathsf{nil}, l, l) \\ \mathsf{app}^+(\mathsf{c}) \multimap & \forall x, l_1, l_2, l_3. \ulcorner \mathsf{app}^-(l_1, l_2, l_3) \urcorner \supset & \\ & & \mathsf{app}^-(\mathsf{c}(x, l_1), l_2, \mathsf{c}(x, l_3)) \end{array}$$

  A goal of the form $\mathsf{app}^-(l_1, l_2, \mathsf{nil})$ will match the first clause for app^-, but not the second (the trigger does not pass the guard). By contrast, a goal of the form $\mathsf{app}^-(l_1, l_2, \mathsf{c}(x, l))$ will match both clauses: the first by binding the variable i to c, the second exactly.

We next describe the encoding of the language of Horn clauses into our target language. As in the transformation based solely on predicate symbols, this encoding has three parts: $\ulcorner G \urcorner$, $\ulcorner\!\ulcorner D \urcorner\!\urcorner$ and $\ulcorner \Gamma \urcorner$ encode goals, clauses and programs respectively. The first two assume the following form:

$$\begin{array}{rcll} \ulcorner a^- \urcorner & = & p^+(c) \multimap a^- & \text{if } a^- = p^-(\vec{t_1}, c(\vec{t}), \vec{t_2}) \\ \ulcorner a^- \urcorner & = & \bigoplus (\exists \vec{y}.\ x = c_i(\vec{y}) \otimes (p^+(c_i) \multimap a^-))_{c_i \in \mathrm{IC}(p^-)} & \\ & & & \text{if } a^- = p^-(\vec{t_1}, x, \vec{t_2}) \\ \ulcorner \top \urcorner & = & \mathbf{1} & \\ \ulcorner G_1 \wedge G_2 \urcorner & = & \ulcorner G_1 \urcorner \otimes \ulcorner G_2 \urcorner & \\ \ulcorner\!\ulcorner G \supset p^-(\vec{t}) \urcorner\!\urcorner & = & \ulcorner G \urcorner \supset p^-(\vec{t}) & \\ \ulcorner\!\ulcorner \forall x.\ D \urcorner\!\urcorner & = & \forall x.\ \ulcorner\!\ulcorner D \urcorner\!\urcorner & \end{array}$$

The two cases for an atomic goal a^- correspond to the situations where the indexing position (highlighted) contains a term starting with a function symbol, c, or is a variable x. In the former case, we simply seed the trigger with the constant associated with c (which we also write c). In the second case we build a disjunction with all elements of $\mathrm{IC}(p^-)$ where p^- is the predicate symbol of a^-. The other cases of the translation are unsurprising.

The translation $\ulcorner \Gamma \urcorner$ of a program relies on the encoding of clauses and proceeds as in our examples. If the indexing position of the head of a clause contains a function symbol c, it is used to seed the indexing guard. If instead it contains a variable, we rely on a variable i to match any trigger for this predicate symbol.

$$\begin{array}{rcl} \ulcorner \cdot \urcorner & = & \forall z.\ z = z \\ \ulcorner \Gamma, D \urcorner & = & \begin{cases} \ulcorner \Gamma \urcorner, p^+(c) \multimap \ulcorner\!\ulcorner D \urcorner\!\urcorner & \text{if } \mathrm{HF}(D) = c \\ \ulcorner \Gamma \urcorner, \forall i.\ p^+(i) \multimap \ulcorner\!\ulcorner D \urcorner\!\urcorner & \text{if } \mathrm{HF}(D) = \bot \end{cases} \\ & & \text{where } p^+ = H(D) \end{array}$$

where $H(D)$ is the index of the head of D, and is defined as in Section 2. The function $\mathrm{HF}(D)$ returns the (indexing constant associated with the) function symbol in the indexing position of clause D, or $\bot$ if this position is contains a variable. Notice that the translation of an empty program installs a clause that interprets $=$ as the identity on terms.

The target language augments the fragment of linear logic seen in Section 2 with various constructs, whose defining rules are shown in Figure 4 (which also contains rules needed in the next section). Specifically, it adds tensor and additional uses of universal quantification in a program position (rules $\otimes_\mathsf{L}$ and $\forall_\mathsf{L}$), and additive conjunction and existential quantification in a goal (rules $\&_\mathsf{R}$ and $\exists_\mathsf{R}$). These additions respect the focusing discipline.

This encoding is sound and complete with respect to the focusing semantics of Horn clauses. The statement of soundness and completeness is identical to the case of Horn clauses, and the proof is a simple extension of what we saw.

$$\frac{}{\Gamma \longrightarrow_\Sigma \boxed{\top}}\,\top_\mathsf{R} \qquad \frac{\Gamma \longrightarrow_\Sigma \boxed{A_1} \quad \Gamma \longrightarrow_\Sigma \boxed{A_2}}{\Gamma \longrightarrow_\Sigma \boxed{A_1 \wedge A_2}}\,\wedge_\mathsf{R}$$

$$\frac{\Gamma, A \longrightarrow_\Sigma \boxed{B}}{\Gamma \longrightarrow_\Sigma \boxed{A \supset B}}\,\supset_\mathsf{R} \qquad \frac{\Gamma \longrightarrow_{\Sigma,x} \boxed{A}}{\Gamma \longrightarrow_\Sigma \boxed{\forall x.\, A}}\,\forall_\mathsf{R}$$

$$\frac{\Gamma, A, \boxed{A} \longrightarrow_\Sigma a^-}{\Gamma, A \longrightarrow_\Sigma \boxed{a^-}}\,\mathsf{atm_R}$$

$$\frac{\Gamma \longrightarrow_\Sigma \boxed{A} \quad \Gamma, \boxed{B} \longrightarrow_\Sigma a^-}{\Gamma, \boxed{A \supset B} \longrightarrow_\Sigma a^-}\,\supset_\mathsf{L} \qquad \frac{\vdash_\Sigma t \quad \Gamma, \boxed{[t/x]A} \longrightarrow_\Sigma a^-}{\Gamma, \boxed{\forall x.\, A} \longrightarrow_\Sigma a^-}\,\forall_\mathsf{L}$$

$$(\textit{no } \top_\mathsf{L}) \qquad \frac{\Gamma, \boxed{A_i} \longrightarrow_\Sigma a^-}{\Gamma, \boxed{A_1 \wedge A_2} \longrightarrow_\Sigma a^-}\,\wedge_{\mathsf{L}i}$$

$$\frac{}{\Gamma, \boxed{a^-} \longrightarrow_\Sigma a^-}\,\mathsf{atm_L}$$

Figure 3: Hereditary Harrop Formulas Provability

The approach to term-based indexing we just presented extends to situations where we want to index an atom based on nested function symbols in the indexing positions (simply associate a constant to each path of interest rather than to just the leading function symbol), and to multiple indexing positions (seed the guard with multiple indexing constants rather than just one). We also expect this approach to generalize to classes of terms broader than first-order terms, although we have not carried out a detailed investigation.

4. BEYOND HORN CLAUSES

The logical unraveling of indexing explored in the last two sections extends well beyond the traditional language of Horn clauses at the basis of Prolog. For example, applying it to backward chaining languages founded on various flavors of linear Horn clauses is a simple adaptation of the above techniques.

In this section, we will instead explore indexing in the context of logic programming languages based on hereditary Harrop formulas [9]. The main difference between hereditary Harrop formulas and Horn clauses is that the former allows universal quantification and embedded implications in a goal position (formulas that can occupy the right-hand side of a sequent). There are two popular presentations of hereditary Harrop formulas. Both have the exact same expressive power, but they require radically different indexing schemes. We call them the minimal and the conjunctive presentations.

Minimal presentation

The minimal presentation of hereditary Harrop formulas is the language freely generated from atomic formulas, implication and universal quantification. There is no distinction between goals and program formulas — we use the word "formula" for both. This language is given by the following grammar.

$$\begin{array}{lrcl} \textit{Formulas:} & A & ::= & a^- \mid A_1 \supset A_2 \mid \forall x.\, A \\ \textit{Programs:} & \Gamma & ::= & \cdot \mid \Gamma, A \end{array}$$

This presentation corresponds exactly to the logic programming language underlying the Twelf system [11].

A focusing definition of provability for this language relies on the same two judgments we saw for Horn clauses, but without a syntactic distinction between goals and program formulas (we will still use these terms depending on whether a formula is on the right or the left of the sequent arrow, respectively). The rules defining provability are included in Figure 3 — simply ignore the rules for conjunction and truth. The main novelty are rules $\supset_\mathsf{R}$ and $\forall_\mathsf{R}$, and the addition of $\mathsf{atm_L}$. Rule $\supset_\mathsf{L}$ gets generalized.

A logic-based treatment of indexing for this presentation of hereditary Harrop formulas is a simple generalization of the method we saw in Sections 2 and 3. Indexing over terms does not change, so we concentrate on predicate symbols. Our target language is similar to the case of Horn clauses, but now allows implications and universal quantifiers in a goal position. The resulting translation is as follows:

$$\begin{array}{rcl} \ulcorner p^-(\vec{t}) \urcorner & = & p^+ \multimap p^-(\vec{t}) \\ \ulcorner A_1 \supset A_2 \urcorner & = & \ulcorner\!\ulcorner A_1 \urcorner\!\urcorner \supset \ulcorner A_2 \urcorner \\ \ulcorner \forall x.\, A \urcorner & = & \forall x.\, \ulcorner A \urcorner \\[4pt] \ulcorner\!\ulcorner p^-(\vec{t}) \urcorner\!\urcorner & = & p^-(\vec{t}) \\ \ulcorner\!\ulcorner A_1 \supset A_2 \urcorner\!\urcorner & = & \ulcorner A_1 \urcorner \supset \ulcorner\!\ulcorner A_2 \urcorner\!\urcorner \\ \ulcorner\!\ulcorner \forall x.\, A \urcorner\!\urcorner & = & \forall x.\, \ulcorner\!\ulcorner A \urcorner\!\urcorner \\[4pt] \ulcorner \cdot \urcorner & = & \cdot \\ \ulcorner \Gamma, A \urcorner & = & \ulcorner \Gamma \urcorner, H(A) \multimap \ulcorner\!\ulcorner A \urcorner\!\urcorner \end{array}$$

where, again, $H(D)$ denotes the index of the head of clause D. It is defined as follows:

$$\begin{cases} H(p^-(\vec{t})) & = \; p^+ \\ H(A_1 \supset A_2) & = \; H(A_2) \\ H(\forall x.\, A) & = \; H(A) \end{cases}$$

This translation is sound and complete with respect to the source language. The statements are identical to the case of Horn clauses (after dropping the distinction between goal and program formulas), and their proof simply adds some cases that are resolved by an immediate application of the induction hypothesis.

Conjunctive presentation

The conjunctive presentation of hereditary Harrop formulas adds conjunction and truth to the mix: goal and program formulas can now freely combine atomic propositions and truth using implication, universal quantification and conjunction. It is therefore defined by the following grammar.

$$\begin{array}{lrcl} \textit{Formulas:} & A & ::= & a^- \mid A_1 \supset A_2 \mid \forall x.\, A \\ & & & \mid \top \mid A_1 \wedge A_2 \\ \textit{Programs:} & \Gamma & ::= & \cdot \mid \Gamma, A \end{array}$$

This presentation of hereditary Harrop formulas is at the core of the logic programming language λProlog [8]. Its linear counterpart underlies the operational semantics of the LLF system [4]. Derivability for this language relies on judgments similar to the minimal presentation, and is defined in full in Figure 3.

The presence of conjunction and truth adds a wrinkle to the task of providing logical foundations to indexing. Program formulas (on the left-hand side) can now have more than one head, as for example $A = a^- \supset (b^- \wedge (c^- \supset d^-))$, which has two heads, b^- and d^-. They can even have no head at all, for example $B = a^- \supset \top$. This means that A needs two indices while B has none.

We can approach indexing for this presentation of hereditary Harrop formulas in two ways. The first is to simply compile it to the minimal presentation examined earlier. The two languages being equivalent in traditional logic, this is always possible. The second is to handle multi-indexing head on. This second approach is unavoidable for linear variants of this language, as found in LLF [4], which are *not* equivalent to the minimal presentation. We will therefore dedicate the rest of this section to examining how to index hereditary Harrop formulas with multiple heads.

Consider again the formula $A = a^- \supset (b^- \wedge (c^- \supset d^-))$. Since it has two heads, b^- and d^-, it can be used in the search of goals of either the form b^- or the form d^-. This means that we must guard it with two indices, b^+ and d^+, with the expectation that we will be able to use the clause only if we supply either b^+ or d^+. One way to achieve this is to combine b^+ and d^+ using additive disjunction, so that the outer layer of the encoding of A looks like

$$(b^+ \oplus d^+) \multimap \textit{<rest of A>}$$

Now, doing just this is not enough: say that our goal is b^-. Then, matching this guard with the trigger b^+ associated with this goal consumes b^+. This has the effect that, although this tells us that this clause contains the goal predicate b^- somewhere, indexing does not help us narrow down the choice further: once we get to the conjunction, it does not provide guidance as to whether to go towards the first conjunct, b^- which will allow us to make progress, or toward the second conjunct, $c^- \supset d^-$ which will lead to failure. This may be a satisfactory solution if we know that our program formulas are simple, with relatively few conjuncts. Then, nothing else is needed to index conjunctive hereditary Harrop formulas.

If instead we want to index the individual conjuncts (and do so recursively for complex program formulas), we need to reassert the trigger associated with the goal, here b^+, and check it again for each conjunct that leads to the head b^-. However, once the guard $b^+ \oplus d^+$ has consumed the trigger (b^+ here), we have lost the information of whether this trigger was b^+ or d^+: we do not know which trigger to reassert. We must therefore refine our strategy. Rather than associating a distinct positive atom (like p^+) to each possible head predicate ($p^-(\vec{t})$ in general), we will employ a single positive predicate, $\mathit{idx}^+(\_)$, whose argument will be a distinct function symbol for each possible head predicate. To avoid symbol proliferation, we will write p for the function symbol associated with predicate symbol p^-. Therefore, the index associated with each predicate with head p^- will now have the form $\mathit{idx}^+(p)$. By doing so, we can capture the current trigger and reassert it once it has passed the guard. The encoding of the formula A above now starts as follows:

$$\forall i.\ \mathit{idx}^+(i)\ \&\ (\mathit{idx}^+(b) \oplus \mathit{idx}^+(d)) \multimap \mathit{idx}^+(i) \otimes \textit{<rest of A>}$$

The current trigger is captured by $\mathit{idx}^+(i)$ in the antecedent of the implication, and bound to the variable i. The additive disjunction forces this same trigger be either b or d. Once it has passed this guard, the trigger is reasserted in the consequent of the implication. While correct in principle, this solution has the disadvantage that the use of $\otimes$ breaks focus, which may lead to trying out a new program formula even though we are not finished with A. We obviate to this by using a nested implication instead of tensor.

These considerations lead to the following encoding:

$$\begin{array}{rcl}
\ulcorner p^-(\vec{t})\urcorner & = & \mathit{idx}^+(p) \multimap p^-(\vec{t}) \\
\ulcorner A_1 \supset A_2\urcorner & = & \ulcorner\!\ulcorner A_1\urcorner\!\urcorner \supset \ulcorner A_2\urcorner \\
\ulcorner \forall x.\, A\urcorner & = & \forall x.\, \ulcorner A\urcorner \\
\ulcorner \top\urcorner & = & \mathbf{1} \\
\ulcorner A_1 \wedge A_2\urcorner & = & \ulcorner A_1\urcorner \otimes \ulcorner A_2\urcorner \\
\\
\ulcorner\!\ulcorner p^-(\vec{t})\urcorner\!\urcorner^i & = & p^-(\vec{t}) \\
\ulcorner\!\ulcorner A_1 \supset A_2\urcorner\!\urcorner^i & = & \ulcorner A_1\urcorner \supset \ulcorner\!\ulcorner A_2\urcorner\!\urcorner^i \\
\ulcorner\!\ulcorner \forall x.\, A\urcorner\!\urcorner^i & = & \forall x.\, \ulcorner\!\ulcorner A\urcorner\!\urcorner^i \\
\ulcorner\!\ulcorner \top\urcorner\!\urcorner^i & = & \top \\
\ulcorner\!\ulcorner A_1 \wedge A_2\urcorner\!\urcorner^i & = & ((\mathit{idx}^+(i) \multimap H(A_1)) \multimap \ulcorner\!\ulcorner A_1\urcorner\!\urcorner^i) \\
 & \& & ((\mathit{idx}^+(i) \multimap H(A_2)) \multimap \ulcorner\!\ulcorner A_2\urcorner\!\urcorner^i) \\
\\
\ulcorner\!\ulcorner A\urcorner\!\urcorner & = & \forall i.\, (\mathit{idx}^+(i)\ \&\ H(A)) \multimap \ulcorner\!\ulcorner A\urcorner\!\urcorner^i \\
\\
\ulcorner\!\ulcorner \cdot\urcorner\!\urcorner & = & \cdot \\
\ulcorner\!\ulcorner \Gamma, A\urcorner\!\urcorner & = & \ulcorner\!\ulcorner \Gamma\urcorner\!\urcorner, \ulcorner\!\ulcorner A\urcorner\!\urcorner
\end{array}$$

where $H(A)$ is the additive disjunction of the indices of all the head predicates in A. It is defined as follows:

$$\begin{cases}
H(p^-(\vec{t})) & = \mathit{idx}^+(p) \\
H(A_1 \supset A_2) & = H(A_2) \\
H(\forall x.\, A) & = H(A) \\
H(\top) & = \mathbf{0} \\
H(A_1 \wedge A_2) & = H(A_1) \oplus H(A_2)
\end{cases}$$

Some additional explanations are in order. The encoding of goals, in the top part, is similar to what we have experienced so far. The top-level translation of program formulas, $\ulcorner\!\ulcorner A\urcorner\!\urcorner$, relies on an inner encoding written $\ulcorner\!\ulcorner A\urcorner\!\urcorner^i$, where i is the variable to which the triggering index from the goal is bound. The cases defining $\ulcorner\!\ulcorner A\urcorner\!\urcorner^i$ for atoms, implication and universal quantification are as in the minimal encoding of hereditary Harrop formulas. This is because these constructs do not alter the set of head predicates made available from their subformula in a program position: they are pass-through. Conjunction instead combines the head predicates of its two conjuncts, as discussed above. Notice how the trigger $\mathit{idx}^+(i)$ is reasserted through an embedded implication to avoid breaking focus. Truth is mapped to the additive unit (also written $\top$), which has no left rule.

The top-level translation of program formulas $\ulcorner\!\ulcorner A\urcorner\!\urcorner$ (also used in embedded goals) is where we capture the trigger, match it against the overall guard, and continue with the rest of the encoding of the program formula. It can be simplified to

$$\ulcorner\!\ulcorner A\urcorner\!\urcorner \quad = \quad H(A) \multimap \ulcorner\!\ulcorner A\urcorner\!\urcorner^i$$

if $i \notin \mathrm{FV}(\ulcorner\!\ulcorner A\urcorner\!\urcorner^i)$, i.e., if A does not contain any conjunction in a program position. Special cases of this situation arise for Horn clauses and minimal hereditary Harrop formulas. Then the resultant encoding is isomorphic to what we have seen for those cases.

It is instructive to look at the result of this encoding on our example formulas. Applying it to $a^- \supset (b^- \wedge (c^- \supset d^-))$ yields the following target formula (we aligned this very formula on the right of its encoding for clarity):

$$\begin{array}{ll}
\forall i.\, \mathit{idx}^{+}(i) \mathbin{\&} (\mathit{idx}^{+}(b) \oplus \mathit{idx}^{+}(d)) \multimap & \\
\quad (\mathit{idx}^{+}(a) \multimap a^{-}) \supset & a^{-} \supset \\
\quad\quad (\quad (\mathit{idx}^{+}(i) \multimap \mathit{idx}^{+}(b)) \multimap b^{-} & \quad (\quad b^{-} \\
\quad\quad\quad \mathbin{\&} (\mathit{idx}^{+}(i) \multimap \mathit{idx}^{+}(d)) \multimap (\mathit{idx}^{+}(c) \multimap c^{-}) & \quad \wedge (c^{-} \\
\quad\quad\quad\quad\quad\quad \supset d^{-})) & \quad\quad \supset d^{-}))
\end{array}$$

The (optimized) encoding of $a^{-} \supset \top$ is as follows:

$$\mathbf{0} \multimap (\mathit{idx}^{+}(a) \multimap a^{-}) \supset \top$$

Notice that this program formula will never be used: there is no right rule for $\mathbf{0}$ and no left rule for $\top$.

Observe that this encoding embed indices much deeper in a (translated) program formula than what we have seen earlier for Horn clauses and for minimal hereditary Harrop formulas. Although it does endow indexing for this language with a logical foundation, its practicality is not as clear.

We conclude this section by formalizing the target language of this encoding:

$$\begin{array}{rrcl}
\textit{Goal formulas:} & \underline{G} & ::= & \mathit{idx}^{+}(p) \multimap a^{-} \mid \underline{D} \supset \underline{G} \mid \forall x.\, \underline{G} \\
 & & & \mid \mathbf{1} \mid \underline{G_1} \otimes \underline{G_2} \\
\textit{Head formulas:} & H & ::= & \mathit{idx}^{+}(p) \mid \mathbf{0} \mid H_1 \oplus H_2 \\
\textit{Program formulas:} & \underline{D} & ::= & a^{-} \mid \underline{G} \supset \underline{D} \mid \forall x.\, \underline{D} \\
 & & & \mid \top \mid T_1 \mathbin{\&} T_2 \\
\textit{Embedded triggers:} & T & ::= & (\mathit{idx}^{+}(i) \multimap H) \multimap D \\
\\
\textit{Programs:} & \underline{\Gamma} & ::= & \cdot \mid \underline{\Gamma}, \forall i.\, (\mathit{idx}^{+}(i) \mathbin{\&} H) \multimap \underline{D} \\
\textit{Active indices:} & \Delta & ::= & \cdot \mid \mathit{idx}^{+}(p)
\end{array}$$

The relevant focused rules of linear logic are displayed in Figure 4. The judgments are similar to those of the other systems in this paper. We write γ and δ for generic formulas from our encoding that can appear in the right-hand side and on the left-hand side of a sequent, respectively. Recall that Δ is either empty or contains one positive atom. The last two rules, $\mathsf{blur_R}$ and $\mathsf{focus_R}$, switch focusing discipline as the polarity of the formula on the right-hand side changes.

The above translation is sound and complete with respect to the conjunctive presentation of hereditary Harrop formulas in Figure 3. The proof is highly technical but not conceptually harder than in our earlier languages.

5. RELATED WORK

We are unaware of prior attempts at giving a logical justification to indexing. However, several authors touched on topics and techniques that are relevant for the present work. Pientka proposed a formalization of the tabling mechanism in Twelf's tabled logic programming engine in terms of a sequent calculus with an explicit indexing context [13]. This is a slightly different notion of "indexing" than the one targeted by this paper, as it is not about the selection of clauses but rather for determining when a goal has been encountered before. It uses sophisticated indexing structures (higher-order substitution trees) that are likely to be beyond the descriptive gamut of focusing. Tabling in proof search in terms of focusing and polarities was explored in [10]. A lock and key protocol similar in spirit to encoding at the heart of this work (see Section 2 for its simplest instance) was proposed in [6], long before linear logic was introduced and focusing discovered. What we achieved here with linearity and positive atoms, these authors achieved using what amounts to double negation and negation-as-failure. Our treatment is however fully declarative while negation-as-failure is not.

6. CONCLUSIONS

In this paper, we have given encodings of various backward chaining programming languages in such a way that program formulas that have no chance of matching a given atomic goal are discarded after limited processing. By doing so, we have proposed a logical foundation to indexing, an implementation technique used to speed up the execution of logic programs. In future work, we plan to expand this technique to situations other than backward logic programming. In particular, we are interested in exploring indexing in forward logic programs (to identify the program formulas activated by the insertion of a new fact or deactivated by the consumption of an existing fact) and in general theorem proving (to select applicable lemmas). We also plan to take the present work to greater depth by refining our treatment of function symbols and program formulas with multiple heads.

Acknowledgments

We are grateful to the anonymous reviewers for pointing us to key related work in the literature and, in general, for helping us improve this paper through valuable comments.

7. REFERENCES

[1] Andreas Abel and Brigitte Pientka, *Higher-order dynamic pattern unification for dependent types and records*, TLCA'11 (Novi Sad, Serbia), Springer-Verlag LNCS 6690, 2011, pp. 10–26.

[2] Iliano Cervesato, *Proof-Theoretic Foundation of Compilation in Logic Programming Languages*, JICSLP'98 (Manchester, UK) (J. Jaffar, ed.), MIT Press, 1998, pp. 115–129.

[3] ———, *An Improved Proof-Theoretic Compilation of Logic Programs*, Theory and Practice of Logic Programming — TPLP **12** (2012), no. 4-5, 639–657.

[4] Iliano Cervesato and Frank Pfenning, *A Linear Logical Framework*, Information & Computation **179** (2002), no. 1, 19–75.

[5] Kaustuv Chaudhuri, Frank Pfenning, and Greg Price, *A logical characterization of forward and backward chaining in the inverse method*, Journal of Automated Reasoning **40** (2008), no. 2–3, 133–177.

[6] Dov Gabbay and Uwe Reyle, *N-Prolog: An extension of Prolog with hypothetical implications, I*, Journal of Logic Programming **1** (1984), no. 4, 319–355.

[7] Chuck Liang and Dale Miller, *Focusing and polarization in intuitionistic logic*, CSL'07, Springer-Verlag LNCS 4646, 2007, pp. 451–465.

[8] Dale Miller and Gopalan Nadathur, *Programming with higher-order logic*, Cambridge University Press, 2012.

[9] Dale Miller, Gopalan Nadathur, Frank Pfenning, and Andre Scedrov, *Uniform proofs as a foundation for logic programming*, Annals of Pure and Applied Logic **51** (1991), 125–157.

[10] Dale Miller and Vivek Nigam, *Incorporating Tables into Proofs*, Proceedings of the 21st International Conference on Computer Science Logic — CSL'07 (Lausanne, Switzerland), Springer-Verlag LNCS 4646, 2007, pp. 466–480.

[11] Frank Pfenning and Carsten Schürmann, *System description: Twelf — a meta-logical framework for*

deductive systems, CADE-16 (Trento, Italy), Springer-Verlag LNAI 1632, 1999, pp. 202–206.

[12] Frank Pfenning and Robert J. Simmons, *Substructural operational semantics as ordered logic programming*, LICS-24 (Los Angeles, CA), 2009, pp. 101–110.

[13] Brigitte Pientka, *Tabled higher-order logic programming*, Ph.D. thesis, Department of Computer Science, Carnegie Mellon University, 2003.

[14] Carsten Schürmann and Frank Pfenning, *A coverage checking algorithm for LF*, TPHOL-16, Springer-Verlag LNCS 2758, 2003, pp. 120–135.

$$\frac{\underline{\Gamma}, \underline{D}; \cdot \longrightarrow_\Sigma \boxed{\underline{G}}}{\underline{\Gamma}; \cdot \longrightarrow_\Sigma \boxed{\underline{D} \supset \underline{G}}}\,\supset_{\mathsf{R}} \qquad \frac{\underline{\Gamma}; \cdot \longrightarrow_{\Sigma,x} \boxed{\underline{G}}}{\underline{\Gamma}; \cdot \longrightarrow_\Sigma \boxed{\forall x.\, \underline{G}}}\,\forall_{\mathsf{R}}$$

$$\frac{\underline{\Gamma}, \delta; p^+, \boxed{\delta} \longrightarrow_\Sigma a^-}{\underline{\Gamma}, \delta; p^+ \longrightarrow_\Sigma \boxed{a^-}}\,\mathsf{atm}_{\mathsf{R}}$$

$$\frac{\underline{\Gamma}; \cdot \longrightarrow_\Sigma \boxed{\underline{G}} \quad \underline{\Gamma}; \Delta, \boxed{\underline{D}} \longrightarrow_\Sigma a^-}{\underline{\Gamma}; \Delta, \boxed{\underline{G} \supset \underline{D}} \longrightarrow_\Sigma a^-}\,\supset_{\mathsf{L}} \qquad (\mathit{no}\ \top_{\mathsf{L}})$$

$$\frac{\vdash_\Sigma t \quad \underline{\Gamma}; \Delta, \boxed{[t/x]\delta} \longrightarrow_\Sigma a^-}{\underline{\Gamma}; \Delta, \boxed{\forall x.\, \delta} \longrightarrow_\Sigma a^-}\,\forall_{\mathsf{L}} \qquad \frac{\underline{\Gamma}; \cdot, \boxed{\underline{T_i}} \longrightarrow_\Sigma a^-}{\underline{\Gamma}; \cdot, \boxed{\underline{T_1} \,\&\, \underline{T_2}} \longrightarrow_\Sigma a^-}\,\wedge_{\mathsf{Li}}$$

$$\frac{}{\underline{\Gamma}; \cdot, \boxed{a^-} \longrightarrow_\Sigma a^-}\,\mathsf{atm}_{\mathsf{L}}$$

$$\frac{\underline{\Gamma}; \delta \longrightarrow_\Sigma \boxed{a^-}}{\underline{\Gamma}; \cdot \longrightarrow_\Sigma \boxed{\delta \multimap a^-}}\,\multimap_{\mathsf{R}} \qquad \frac{\underline{\Gamma}; p^+ \longrightarrow_\Sigma \boxed{\gamma} \quad \underline{\Gamma}; \boxed{\underline{D}} \longrightarrow_\Sigma a^-}{\underline{\Gamma}; p^+, \boxed{\gamma \multimap \underline{D}} \longrightarrow_\Sigma a^-}\,\multimap_{\mathsf{L}}$$

$$\frac{\underline{\Gamma}; p^+ \longrightarrow_\Sigma \boxed{T_1} \quad \underline{\Gamma}; p^+ \longrightarrow_\Sigma \boxed{T_2}}{\underline{\Gamma}; p^+ \longrightarrow_\Sigma \boxed{T_1 \,\&\, T_2}}\,\&_{\mathsf{R}} \qquad \frac{\underline{\Gamma}; p^+ \longrightarrow_\Sigma \boxed{H_i}}{\underline{\Gamma}; p^+ \longrightarrow_\Sigma \boxed{H_1 \oplus H_2}}\,\oplus_{\mathsf{R_i}}$$

$$\frac{\vdash_\Sigma t \quad \underline{\Gamma}; \cdot \longrightarrow_\Sigma \boxed{[t/x]A}}{\underline{\Gamma}; \cdot \longrightarrow_\Sigma \boxed{\exists x.\, A}}\,\exists_{\mathsf{R}}$$

$$\frac{\underline{\Gamma}; p^+ \longrightarrow_\Sigma \boxed{A^-}}{\underline{\Gamma}; p^+ \longrightarrow_\Sigma \boxed{A^-}}\,\mathsf{blur}_{\mathsf{R}} \qquad \frac{\underline{\Gamma}; p^+ \longrightarrow_\Sigma \boxed{A^+}}{\underline{\Gamma}; p^+ \longrightarrow_\Sigma \boxed{A^+}}\,\mathsf{focus}_{\mathsf{R}}$$

(δ and γ are generic formulas from the encoding in Section 4)

Figure 4: Additional Linear Logic Rules