

Indeed: Interactive Deduction on Horn Clause Theories

Oscar Olmedo-Aguirre¹ and Guillermo Morales-Luna^{2*}

¹ Computer Science Section, CINVESTAV-IPN,
Av. I. P. N. 2508, 07300 Mexico City, MEXICO,
`oolmedo@cs.cinvestav.mx`

² Computer Science Section, CINVESTAV-IPN,
Av. I. P. N. 2508, 07300 Mexico City, MEXICO,
`gmorales@cs.cinvestav.mx`

Abstract. We introduce the declarative programming language **Indeed** that uses both deduction and interaction through multi-agent system applications. The language design is addressed by providing a uniform programming model that combines two refinements of resolution along with some control strategies to introduce state-based descriptions. We show that the logical calculus in which the computational model is based is sound and complete. Finally, we compare our approach to others proposed for coupling interaction with automated deduction.

Keywords. Logic programming, interaction, automated theorem proving, Horn clause theories.

1 Introduction

Artificial intelligence, largely based on formal logic and automated reasoning systems, has become increasingly more interactive. As indicated by Wegner [10], dynamic acquisition of interactive knowledge is fundamental to diminish the complexity of interactive tasks, to better their performance, and to better the expressiveness of their modeling. On the other hand, there is a fundamental distinction between deduction and interaction. In high order logics provers, as HOL [4] and, particularly, Isabelle [6], interaction is meant as a direct user guidance to construct a proof. The kind of interaction we have in mind is closer to the notion shown in [3]: different agents scan a common work-place to pursue with their common goal, e.g. to prove the current theorem.

Models of algorithmic computation, like automated theorem provers and Turing machines, are characterized by their closed and monolithic approach. Their simple observable behavior comprises a three-stage process of interactive input, closed data processing, and interactive output. Nonetheless, modern applications involving collaboration, communication and coordination, require richer behaviors that cannot solely be obtained from algorithmic computation.

* Partially supported by Programa de Ingeniería Molecular, Instituto Mexicano del Petróleo.

In this work, we propose a logic programming language that extends a resolution-based theorem prover with interaction. The computational model comprises SLD-resolution and UR-resolution, to describe respectively stateless deduction and state-based transitions. The coordination model consists of a transactional global memory of ground facts along with a strategy for the theorem prover to control program execution by syntactically guided rule selection. In addition, the set of support restriction strategy coordinates the input and output of facts with the shared memory, maintaining the coherence of the current state of the computing agent.

Let us briefly explore other approaches that can be compared with ours: resolution theorem provers, constraint logic programming and coordination logic programming.

The resolution-based theorem prover *OTTER* [11, 12] comprises a number of refinements of resolution along with a set of control strategies to prune the explosive generation of intermediate clauses. However, *OTTER* does not account for interaction. The set of all instantaneous descriptions essentially corresponds to the set of support strategy. In *OTTER*, a clause is selected and removed from the set of support to produce a new set of clauses deduced from the axioms of the theory. Then, after simplifying a new clause by demodulation and possibly discarding it by either weighting, backward or forward subsumption, the new clause is placed back to the set of support.

Concurrent Constraint Programming (CCP) [7] proposes a programming model centered on the notion of constraint store that is accessed through the basic operations 'blocking *ask*' and 'atomic *tell*'. Blocking *ask*(*c*) corresponds to the logical entailment of constraint *c* from the contents of the constraint store: the operation blocks if there is not an enough strong valuation to decide on *c*. In this respect, the blocking mechanism is similar to the one used in *Indeed* to obtain the set of ground facts that match with the left-hand side of some rule. Besides, the constraint store shares some similarities with the global memory of ground facts. However, operation *tell*(*c*) is more restrictive than placing ground atoms in the global memory because constraint *c* must be logically consistent with the constraint store.

Finally, *Extended Shared Prolog* (ESP) [3] is a language for modeling rule-based software processes for distributed environments. ESP is based in the PoliS coordination model that extends Linda with multiple tuple spaces. The language design seeks for combining the PoliS mechanisms for coordinating distribution with the logic programming Prolog. Coordination takes place in ESP through a named multiset of passive and active tuples. They correspond to the global memory of facts in *Indeed*, although no further distinction between passive and active ground facts is made. ESP also extends Linda by using unification-based communication and backtracking to control program execution. In relation to the theoretical foundations, ESP has not a clean integration of interaction and deduction as suggested by *Indeed* in which the coordination component is given by providing a particular operational interpretation of the memory of facts.

The paper is organized as follows. First we illustrate the forward and backward search schemes that arise from our model with a programming example. Next, we introduce the declarative language: the syntax and the declarative semantics of the programming language.

2 A programming example

Dynamic systems can generally be decomposed into a collection of computing agents with local state. The distributed state of the entire system corresponds to the set of local states of each individual agent, whereas the specification of the system behavior is described by the axioms of logical theories. As agents perceive the surrounding environment through *sensors* and act upon it through *effectors*, their interaction can effectively be decoupled by a coordination medium consisting of a multiset of ground facts. By abstracting away interaction from deduction, the inherently complex operational details of sensors and effectors become irrelevant.

As an example, consider the problem of parsing simple arithmetic expressions. The compiler uses the context free grammar (CFG):

$$E \rightarrow 0 \mid 1 \mid (E) \mid E + E \mid E \times E$$

where non-terminal E stands for “expression”. Ground atoms are used to represent the tokens forming the input expression and to represent well-formed sub-expressions as well.

Table 1 shows theory *Natural* for the natural numbers written in **Indeed**. This theory uses *backward rules* that have the general form $p \Leftarrow p_1, \dots, p_n$ with $n \geq 0$. The logical propositions of the theory are built upon infix predicates $=$, $<$, and \leq , whose recursive definitions are given by pure Prolog clauses *N1* to *N5*. *Natural* represents the computational component of the interactive parser.

theory <i>Natural</i>	
axioms	
[N1]	$0 + y = y \Leftarrow .$
[N2]	$(x + 1) + y = (x + y) + 1 \Leftarrow .$
[N3]	$0 \leq y \Leftarrow .$
[N4]	$(x + 1) \leq (y + 1) \Leftarrow x \leq y.$
[N5]	$x < y \Leftarrow (x + 1) \leq y.$
end	

Table 1. Natural numbers using backward rules.

Table 2 shows the theory *Parser* that extends *Natural* written in **Indeed**. This theory uses *forward rules* that have the general form $p_1, \dots, p_n \Rightarrow p$ with $n \geq 0$. The rules of *Parser* define a bottom-up parser for simple arithmetic expressions

whose syntactic entities are represented by ground atoms. $T(n, x)$ asserts that token x occurs at position n while $E(n1, n2)$, with $n1 \leq n2$, asserts that the sequence of tokens from $n1$ to $n2$ is a well-formed arithmetic expression. It is defined by self-explanatory rules in accordance with the given CFG.

theory <i>Parser</i>
extends <i>Natural</i>
axioms
[P1] $T(n, '0') \Rightarrow E(n, n)$.
[P2] $T(n, '1') \Rightarrow E(n, n)$.
[P3] $T(n1, '('), E(n2, n3), T(n4, ')')$ $\quad \quad n1 + 1 = n2, n2 \leq n3, n3 + 1 = n4$ $\quad \Rightarrow E(n1, n4)$.
[P4] $E(n1, n2), T(n3, '+'), E(n4, n5)$ $\quad \quad n1 \leq n2, n2 + 1 = n3, n3 + 1 = n4, n4 \leq n5$ $\quad \Rightarrow E(n1, n5)$.
[P5] $E(n1, n2), T(n3, '\times'), E(n4, n5)$ $\quad \quad n1 \leq n2, n2 + 1 = n3, n3 + 1 = n4, n4 \leq n5$ $\quad \Rightarrow E(n1, n5)$.
end

Table 2. Bottom-up parser for arithmetic expressions.

Table 3 sketches the interaction of the parser that takes place in the common memory with sensors and effectors, dealing with the rather trivial, although illustrative, string: $'(1+0) \times 1'$. Atoms based on predicate T are placed in the memory by sensors and displayed from left to right as they are produced. As soon as the atoms occurring on the left-hand side of some rule of *Parser* become available, the parser removes them and places the atom occurring on the right-hand side of the rule. An effector may take the last produced atom to inform the editor that the analyzed expression is well-formed.

A transactional memory ensures the 'all-or-nothing' property in the shared resource concurrent access. Intuitively, the interaction parser proceeds as follows:

1. The initial agent's state is given by ground atoms. As no inputs have been detected from sensors, no rules apply and nothing can be deduced.
2. Eventually, after detecting some activity, sensors place in the shared memory the initial readings as ground atoms.
3. The interactive component determines which responses are necessary by selecting an appropriate forward rule whose left-hand side have a match with the contents of the shared memory.
4. Once a forward rule is selected, the reasoning component attempts to prove that its guarding constraints are satisfied.
5. Whenever the drawn conclusions hold, responses are produced by placing the ground atoms of the consequent part into the shared memory.

6. However, if the drawn conclusions are false, another forward rule, if any, is selected. If there is not a successful forward rule available, the agent waits for the reading of appropriate ground atoms obtained from the sensors.
7. Computation continues until no agent can apply a forward rule.

3 Indeed formal description

3.1 Inference rules

The names signature of the language comprises a set C of *constructor names* and a set X of *variable names*. The set $T(X)$ of *terms with variables* is the minimal set containing $C \cup X$, closed under composition of a constructor with a term sequence. The set T of *ground terms* consists of the terms with no variables. The set $A(X)$ consists of *atoms*, e.g. compositions of predicate symbols with term sequences. The set A of *ground atoms* consists of all atoms with no variables. A *clause* is a disjunction of *literals*, i.e. atoms or negated atoms, and is represented as a set $P = \{p_1, \dots, p_n\}$. An *unit clause* contains only one atom. A *positive*

clause contains no negated atoms. A *negative clause* contains no positive atoms. A *definite clause* contains at most a positive atom. A *goal* consists only of negative atoms and the set of goals is denoted by $G(X)$. A *guarded goal* has the form $P \mid G$ in which the atoms of P are defined by forward rules whereas the atoms of G are defined by backward rules. The set of all guarded atoms is denoted by $GG(X)$. A *substitution* is a map $\sigma : X \rightarrow T(X)$ and it admits a natural extension to terms $\sigma : T(X) \rightarrow T(X)$.

The theory Ax consists of all backward and forward rules. The theory Bx consists only of the backward rules in Ax . Both, backward and forward rules, have the same declarative reading: $p \Leftarrow P$ and $P \Rightarrow p$ are read *p holds if P holds*. In any case, p and P correspond to the *antecedent* and *consequent* parts of the rule. Nonetheless, the operational interpretations of the rule types are remarkably different. Backward rules have a goal directed control strategy, whereas forward rules are driven by unit clauses representing the currently known set of facts. With respect to unit clauses, their procedural interpretations coincide. Indeed, the proposed notation maintains the declarative meaning of clauses and makes explicit both the resolution method and the control strategy to be used. The forward rules denotation of the usual Horn clause logic is extending the expressiveness of classical logic programming languages. Let $Th(Ax)$ and $Th(Bx)$ be the sets of *deduced clauses* from Ax and Bx respectively. Deduction is determined by the application of the *inference rules* shown in Table 4

<i>Instantiation rules</i>	
$BI : \frac{p \Leftarrow P \in Bx \quad \sigma : X \rightarrow T(X)}{p\sigma \Leftarrow P\sigma \in Th(Bx)}$	$FI : \frac{P \mid G \Rightarrow p \in Ax \quad \sigma : X \rightarrow T(X)}{P\sigma \mid G\sigma \Rightarrow p\sigma \in Th(Ax)}$
<i>Resolution rules</i>	
$BR : \frac{p \Leftarrow P \cup \{q\} \in Th(Bx) \quad q \Leftarrow Q \in Th(Bx)}{p \Leftarrow P \cup Q \in Th(Bx)}$	$FR : \frac{p_1 \cdots p_n \mid G \Rightarrow p \in Th(Ax) \quad \Leftarrow G \in Th(Bx) \quad \Rightarrow p_i \in Th(Ax) \ \forall i \leq n}{\Rightarrow p \in Th(Ax)}$

Table 4. Inference rules.

Backward rules roughly correspond to the procedural interpretation of SLD-resolution, while forward rules can be related to UR-resolution. Both resolutions are known to be sound and refutation complete on definite clauses. Furthermore they can be combined with other strategies such as the set of support.

4 Interpretations, soundness and completeness

Let H_{Ax} be Herbrand universe of Ax . The Herbrand base B_{Ax} for the definite program Ax is the set of all ground atoms formed using predicate symbols com-

posed with ground terms at the Herbrand universe. The relation $\mathbf{A} \models C\sigma$ among a Herbrand algebra \mathbf{A} , a substitution σ and a clause C is canonical: C is true when its variables are substituted according to σ . A clause C is true in the interpretation \mathbf{A} , written $\mathbf{A} \models C$, if C is true for every substitution. A clause C is not true in the interpretation, $\mathbf{A} \not\models C$, when C is not true for some substitution. When C is an unsatisfiable goal, we may write $\mathbf{A} \models \Leftarrow C$ instead of $\mathbf{A} \not\models C$. Also:

- If $p(t_1, \dots, t_n)$ is an atom, then $\mathbf{A} \models p(t_1, \dots, t_n)\sigma$ iff $(t_1\sigma, \dots, t_n\sigma) \in p_{\mathbf{A}}$.
- $\mathbf{A} \models \{p_1, \dots, p_n\}\sigma$ iff $\mathbf{A} \models p_i\sigma$ for all $i = 1, \dots, n$.
- $\mathbf{A} \models (p \Leftarrow P)\sigma$ iff $\mathbf{A} \models P\sigma$ implies $\mathbf{A} \models p\sigma$.
- In particular, $\mathbf{A} \models (p \Leftarrow)\sigma$ iff $\mathbf{A} \models p\sigma$.
- $\mathbf{A} \not\models \Leftarrow$.

For a set S of clauses, \mathbf{A} models S if \mathbf{A} is an interpretation for each clause in S . Given two set of clauses S and T , S *semantically implies* T , $S \models T$, if every model of S is also a model of T . The following is immediate:

Lemma 1. *Let C and S be sets of definite clauses and G a definite goal. Then,*

1. $C \models C\sigma$ for any substitution σ
2. $S \cup \{C\} \models C$
3. $S \models S'$ and $S' \models S''$ imply $S \models S''$
4. $S \models C$ implies $S \models C'$ if $C \models C'$

Let us consider the problem to decide whether $\text{Ax} \models C$. Let \mathcal{I} be a Herbrand interpretation for Ax . The monotonic map $T_{\text{Ax}} : 2^{\mathbf{B}_{\text{Ax}}} \rightarrow 2^{\mathbf{B}_{\text{Ax}}}$, $\mathcal{I} \mapsto T_{\text{Ax}}(\mathcal{I}) \triangleq \{p\sigma \in \mathbf{B}_{\text{Ax}} \mid P \Rightarrow p \in \text{Ax}, P\sigma \subseteq \mathcal{I}, \sigma : \text{X} \rightarrow \text{T}\}$ progressively determines the set of ground atoms that are logical consequence from the theory. Namely, $T_{\text{Ax}}(\emptyset) = \{p\sigma \mid (\Rightarrow p) \in \text{Ax} \text{ and } \sigma \text{ a ground substitution}\}$ corresponds to the set of all ground facts of the theory. Let:

$$T_{\text{Ax}}^0 \triangleq \emptyset ; \quad T_{\text{Ax}}^n \triangleq T(T_{\text{Ax}}^{n-1}) \text{ if } n > 0 ; \quad T_{\text{Ax}}^\omega \triangleq \bigcup_{n=0} T_{\text{Ax}}^n.$$

We may identify the least Herbrand model with the minimal model $\mathbf{A} = T_{\text{Ax}}^\omega$. The *success set* consists of all ground atoms refutable in the backward theory.

4.1 Soundness and completeness of forward deduction

The following propositions are proved straightforwardly:

Proposition 1. *Forward resolution preserves consistency. I.e.:*

$$(\text{Ax} \models Q \mid G \Rightarrow p \ \& \ \text{Ax} \models G\sigma \ \& \ \text{Ax} \models q\sigma \ \forall q \in Q) \Rightarrow \text{Ax} \models p\sigma$$

Proposition 2 (Soundness of forward resolution as a deduction calculus). *Any ground atom q derived from Ax is a semantic consequence of Ax .*

Also, the following proposition holds:

Proposition 3 (Completeness of forward resolution as a deduction calculus). *If a positive ground atom p has a Herbrand model, then p can be derived from Ax by forward resolution.*

Proof. If the positive ground atom p has a Herbrand model then one of the following relations holds:

1. $Ax \models p$
2. T_{Ax}^ω is a model for $\Rightarrow p$
3. By definition of T_{Ax} , there exists a minimal integer n such that $\Rightarrow q\sigma \in T_{Ax}^n$. Then, there exists a rule $Q \mid G \Rightarrow q$ and a ground substitution σ such that, either $Q\sigma \mid G\sigma \subseteq \emptyset$ if $n = 0$ or $G\sigma : Q\sigma \subseteq T_{Ax}^{n-1}$ if $n > 0$, with $p = q\sigma$

In any case, it follows directly that $Ax \vdash \Rightarrow p$.

4.2 Computational model

In this section we will construct a computational model for the previously defined logic. Let $ID = GG(X) \times (X \rightarrow T(X))$ consist of *instantaneous descriptions*, i.e. pairs id consisting of a guarded goal and a substitution instantiating the variables appearing at the goal. ID can be provided of two *transition* relations $\triangleleft, \triangleright \subset \mathcal{P}(ID) \times \mathcal{P}(ID)$ between sets of instantaneous descriptions, each one defining, respectively, the backward and forward computation relations. Transition relations are determined by the rules shown in Table 5.

<i>Backward computation</i>	
$\frac{p \Leftarrow P \in Bx}{(\Leftarrow G \cup \{p\sigma_2\}, \sigma_1) \triangleleft (\Leftarrow G\sigma_2 \cup P\sigma_2, \sigma_1\sigma_2)}$	
<hr/>	
<i>Forward computation</i>	
$\frac{p_1 \cdots p_n \mid G \Rightarrow p \in Ax}{(\Leftarrow G, \sigma) \triangleleft^* (\Leftarrow, \sigma)}$	
$\frac{I \cup \{(p_1\sigma, \sigma_1), \dots, (p_n\sigma, \sigma_n)\}}{I \cup \{(\Rightarrow p\sigma, \sigma_1 \cdots \sigma_n\sigma)\}}$	

Table 5. Rules for relation “transition”.

Given an instantaneous description $(\Leftarrow \{p\sigma_2\} \cup G, \sigma_1)$, the first transition applies backward resolution over corresponding instances σ_2 of the goal $\{p\} \cup G$ and the rule $p \Leftarrow P$. If more than one backward rule can be applied, just one candidate rules is non-deterministically selected. The second transition applies forward resolution over a set of ground atoms $\{(\Rightarrow p_1\sigma, \sigma_1), \dots, (\Rightarrow p_n\sigma, \sigma_n)\}$ for some $n > 0$. The transition requires a suitable instance of rule $p_1, \dots, p_n \mid G \Rightarrow p$ under some substitution σ such that $p_i\sigma = q_i\sigma$ for each $i \in \{1, \dots, n\}$. Then, a

new instantaneous description ($\Rightarrow p\sigma, \sigma_1 \cdots \sigma_n \sigma$) is obtained if the guarding condition $G\sigma$ has a refutation with σ as the computed answer. If the guard $G\sigma$ fails, another forward rule is selected non-deterministically. Note that σ must lead to a ground goal $G\sigma$. The substitution $\sigma_1 \cdots \sigma_n \sigma$ enables a form of unification-based communication among the computing agents.

Let \hookrightarrow be the reflexive-transitive closure of the union of \triangleright and \triangleleft , $\hookrightarrow^* = (\triangleright \cup \triangleleft)^*$. Then, the correctness of the computational model can be stated as follows:

Proposition 4. *For any Q, G, p :*

$$Ax \models Q \mid G \Rightarrow p \Leftrightarrow \forall \sigma : (Ax \cup Q \cup G, \sigma) \hookrightarrow^* (p, \sigma).$$

We have implemented the programming model in a prototype system written in Prolog consisting of a compiler and a small run-time library. The main difficulty in this implementation lies in the inherent complexity posed of selecting appropriate forward rules. A forward rule with n antecedents, each one having a set of k atoms realizes a match in time complexity $O(k^n)$ in the worst case. Fortunately, most applications typically have rules with no more than two antecedents. Our architecture adopts an event-driven approach that reduces the complexity in one order, i.e. $O(k^{n-1})$. Although the results are encouraging, we believe that to take full-advantage of this approach, a further improvement in the expressiveness of the language constructs is still necessary.

5 Conclusions

In this paper, we have addressed the problem of coupling interaction in resolution theorem provers. Our experimental programming language **Indeed** has been designed by distinguishing between state-based descriptions using forward rules and stateless deduction using backward rules. The distinction is important and convenient as the programming model allow us to combine backward and forward rule chaining in a single model. We have shown that our calculus is sound and complete in the limit case in which no interaction occurs.

References

1. M. Belmesk, Z. Habbas, and P. Jorrand A Process Algebra over the Herbrand Universe: Application to Parallelism in Automated Deduction. In B. Fronhofer and G. Waghston. *Parallelization of inference Systems*. LNAI 590. Springer-Verlag. 1990.
2. N. Carrierio and D. Gelernter, Linda in Context, *CACM*, 32(4):444-458, Apr 1989.
3. P. Ciancarini Coordinating Rule-Based Software Processes with ESP, *ACM Trans. on Software Engineering and Methodology*, 2(3):203-227, July, 1993.
4. Gordon, M. J. C., and Melham, T. F. *Introduction to HOL: A Theorem Proving Environment for Higher-Order Logic*. Cambridge University Press, 1993.
5. J.W. Lloyd, *Foundations of Logic Programming*, Springer-Verlag, 1987.
6. Nipkow, T. , *Isabelle HOL: The Tutorial*, <http://www.in.tum.de/~nipkow/>, 1998
7. V.A. Saraswat Concurrent Constraint Programming. *Records of 17th ACM Symposium on Principles of Programming Languages*, 232-245. San Francisco, CA. 1990.

8. S. Tahar and P. Curzon, Comparing HOL and MDG: A Case Study on the Verification of an ATM Switch Fabric. *Nordic Journal of Computing*, pp. 372-402, 6(4), Winter 1999.
9. R.D. Tennet, *Semantics of Programming Languages*, Prentice Hall Intl., 1991.
10. P. Wegner, Interactive Software Technology, *CRC Handbook of Computer Science and Engineering*, May 1996.
11. L. Wos, R. Overbeek, E. Lusk, and J. Boyle, *Automated Reasoning. Introduction and Applications*, McGraw-Hill, Inc., 1992.
12. L. Wos and G. Pieper, *A Fascinating Country in the World of Computing: Your Guide to Automated Reasoning*, World Scientific Publishing Co., 1999.