From:MAICS-99 Proceedings. Copyright © 1999, AAAI (www.aaai.org). All rights reserved.

INDED: A Symbiotic System of Induction and Deduction *

Jennifer Seitzer†
Computer Science Department
University of Dayton
300 College Park
Dayton, Ohio 45469-2160
seitzer@cps.udayton.edu

Abstract

We present an implementation of stable inductive logic programming (stable-ILP) [Sei97], a cross-disciplinary concept bridging machine learning and non-monotonic reasoning. In a deductive capacity, stable models give meaning to logic programs containing negative assertions and cycles of dependencies. In stable-ILP, we employ these models to represent the current state specified by (possibly) negative extensional and intensional (EDB and IDB) database rules. Additionally, the computed state then serves as the domain background knowledge for a top-down ILP learner.

In this paper, we discuss the architecture of the two constituent computation engines and their symbiotic interaction in computer system **INDED** (pronounced "indeed"). We introduce the notion of *negation as failure-to-learn* and provide a real world source of negatively recursive rules (those of the form $p \leftarrow \neg p$) by explicating scenarios that foster induction of such rules. Last, we briefly mention current work using **INDED** in data mining.

Keywords: inductive logic programming, logic programming, data mining.

Introduction

A logic program is a set of logic formulas grouped together to computationally solve a problem or perform a task. Logic programs serve as mechanisms of knowledge representation which are useful in expert system, deductive database, and truth maintenance system design. They are powerful vehicles of data augmentation that enable artificial intelligence computer systems to increase information using the techniques of logical deduction. Because of this ubiquitousness, logic programs also serve as useful dialects of discovered knowledge in data mining and knowledge discovery systems. That is, they provide a convenient language for new information acquired by induction (machine learning). In fact, logic programs are the output of the discovery systems of Inductive Logic Programming (ILP), one type of machine learning.

A semantics of a propositional logic program can be thought of as a function which produces the set of facts logically implied by the program, and therefore, states which propositions mentioned in the program are considered **true** and which are considered **false**. The stable semantics [GL90] and its three-valued approximation, the well-founded semantics [VRS91], are able to interpret logic programs with rules containing negative literals and (positive and negative) cycles of dependencies formed amongst its literals [1]. Stable semantics provide total truth assignments to all propositions in the knowledge base, but, in general, are computationally expensive (stable model computation is an NP-hard problem). A well-founded model, however, can be computed in quadratic time, but is not always total. Thus, some propositions may be left unassigned.

In this work, we apply these semantics to inductive logic programming (ILP) by utilizing domain (background) knowledge in the form of normal logic programs with explicit negation and cycles [Sei97]. The semantics are applied to the background logic program to acquire facts from which new hypotheses can be constructed. Moreover, by iterating between the deduction and induction engines, we are able to learn, or induce, both positively and negatively recursive rules as well as rule sets forming larger cycles of dependencies. We refer to this concept of applying the stable semantics to ILP as stable-ILP.

One of the most exciting discoveries in this work was finding a source of negatively recursive rules (those of the form $p \leftarrow \neg p$). These rules preclude the existence of stable models in normal logic programs and have stumped logic programming researchers as to whether or not they

*Copyright @ 1999, American Association for Artificial Intelligence (www.aaai.org). All rights reserved.

†This work is supported by grant 9806184 of the National Science Foundation.

[1] Although the formal definitions of these semantics are cited above, for this paper, we can intuitively accept stable and well-founded models as those sets of propositions that are generated by transitively applying modus ponens to rules.

represent any real-world state or scenario. Such rules exist in deliberately syntactically derived examples such as in Van Gelder's famous example [Van93], yet there is much skepticism as to whether or not any real-world scenarios render such rules. We show that by juxtaposing contexts when employing induction, such anomalous rules can be induced.

Inductive Logic Programming

Inductive logic programming (ILP) is a new research area in artificial intelligence which attempts to attain some of the goals of machine learning while using the techniques, language, and methodologies of logic programming. Some of the areas to which ILP has been applied are data mining, knowledge acquisition, and scientific discovery [LD94]. The goal of an inductive logic programming system is to output a rule which *covers* (entails) an entire set of positive observations, or examples, and *excludes* or *does not cover* a set of negative examples [Mug92]. This rule is constructed using a set of known facts and rules, knowledge, called domain or *background* knowledge. In essence, the ILP objective is to synthesize a logic program, or at least part of a logic program using examples, background knowledge, and an entailment relation. The following definitions are from [LD94].

Definition 2.1 (coverage) *Given background knowledge $\mathcal{B}$, hypothesis $\mathcal{H}$, and example set $\mathcal{E}$, hypothesis $\mathcal{H}$* covers *example $e \in \mathcal{E}$ with respect to $\mathcal{B}$ if $\mathcal{B} \cup \mathcal{H} \models e$.*

Definition 2.2 (complete) *A hypothesis $\mathcal{H}$ is* **complete** *with respect to background $\mathcal{B}$ and examples $\mathcal{E}$ if all positive examples are covered, i.e., if for all $e \in \mathcal{E}^+$, $\mathcal{B} \cup \mathcal{H} \models e$.*

Definition 2.3 (consistent) *A hypothesis $\mathcal{H}$ is* **consistent** *with respect to background $\mathcal{B}$ and examples $\mathcal{E}$ if no negative examples are covered, i.e., if for all $e \in \mathcal{E}^-$, $\mathcal{B} \cup \mathcal{H} \not\models e$.*

Definition 2.4 (Formal Problem Statement) *Let $\mathcal{E}$ be a set of training examples consisting of* **true** *$\mathcal{E}^+$ and* **false** *$\mathcal{E}^-$ ground facts of an unknown (target) predicate $\mathcal{T}$. Let $\mathcal{L}$ be a description language specifying syntactic restrictions on the definition of predicate $\mathcal{T}$. Let $\mathcal{B}$ be background knowledge defining predicates q_i which may be used in the definition of $\mathcal{T}$ and which provide additional information about the arguments of the examples of predicate $\mathcal{T}$. The ILP problem is to produce a definition $\mathcal{H}$ for $\mathcal{T}$, expressed in $\mathcal{L}$, such that $\mathcal{H}$ is complete and consistent with respect to the examples $\mathcal{E}$ and background knowledge $\mathcal{B}$. [LD94]*

Along with extending the language of $\mathcal{B}$, we also define the notion of coverage in terms of the stable semantics. The hypothesis language, at this point, shall remain as (possibly) recursive datalog clauses, where "datalog" means function-free except for constants.

Stable-ILP Framework

The following framework allowed us to integrate the functionality of a bottom-up (forward reasoning) non-monotonic deductive system with a top-down, ILP learning system.

Definition 2.5 (Stable Coverage) *Let* P *be the logic program $\mathcal{B} \cup \mathcal{H}$. We call the set of all propositions assigned* **true** *by the well-founded model of logic program* P, *$(\mathcal{B} \cup \mathcal{H})_{WF}$. An example e is* well-foundedly covered *if $e \in (\mathcal{B} \cup \mathcal{H})_{WF}$. Likewise, we call the set of all propositions assigned* **true** *by any stable model of* P, *$(\mathcal{B} \cup \mathcal{H})_S$. An example e is* stably covered *if $e \in (\mathcal{B} \cup \mathcal{H})_S$.*

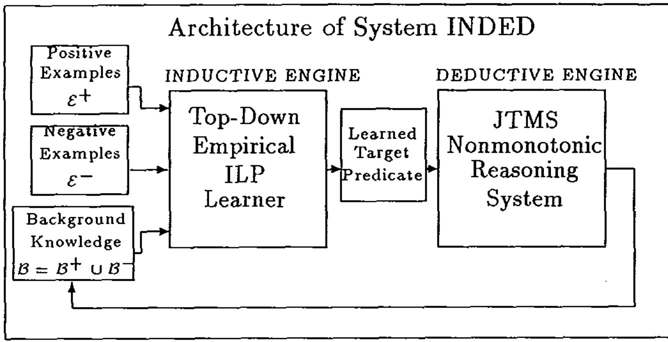
The following actions reflect the iterative, synergistic behavior of a stable-ILP system. The steps of one iteration are:

1. Compute the ground instantiation of logic program $\mathcal{B}$ called $\mathcal{B}_{\mathcal{G}}$.
2. Compute the well-founded and/or stable models of $\mathcal{B}_{\mathcal{G}}$ called $\mathcal{B}_{\mathcal{WF}}$ and $\mathcal{B}_{\mathcal{S}}$ respectively
3. Induce hypothesis $\mathcal{H}$ using $\mathcal{B}_{\mathcal{WF}}$ and/or $\mathcal{B}_{\mathcal{S}}$. While ascertaining whether or not an example is *covered* by a prospective hypothesis $\mathcal{H}$, use the definition of *stably covered.*
4. Augment IDB with newly learned intensional rule(s).

Framework Implementation

System **INDED** (pronounced "indeed") is the author's realization of stable-ILP. It is comprised of two main computation engines. The deduction engine is a bottom-up reasoning system that computes the current state by generating a stable model, if there is one, of the current ground instantiation, and by generating the well-founded model, if there is no stable model. This deduction engine is, in essence, a justification truth maintenance system [Doy79] which accommodates non-monotonic updates in the forms of positive or negative facts. It also accommodates manually entered augmentations in the form of intensional and extensional rules.

The induction engine, using the current state created by the deduction engine, along with positive examples $\mathcal{E}^+$ and negative examples $\mathcal{E}^-$, induces an intensional rule(s) which is then added to the deductive engine's intensional rule set. This iterative, symbiotic behavior mimics theory revision systems [MT92]. The following diagram illustrates the constituents of **INDED** and their symbiotic interaction.



Recently, there has been other research combining induction and deduction [GCM95] [GC96], but the employment of negation in the intensional domain knowledge and the generation of negative facts by the deduction engine to be used by the induction engine to learn (possibly) recursive rules is new.

The algorithms utilized in both the deductive and inductive engines are heavily dependent on the chosen data structures. The data structures are implemented as a labyrinth of linked lists of aggregate classes in C++.

The Deductive Engine

As in most logic programming systems, we assume the existence of an initial extensional knowledge base (EDB) consisting of ground facts (i.e., facts with all constants and no variables). We also assume an initial set of rules with variables that reflect information about the knowledge base called the intensional knowledge base (IDB). Jointly, the EDB and IDB serve as a start state from which computational deduction can commence.

Hypergraph Representation

The deductive reasoning system represents the ground instantiation $\mathbf{P}$ of the combined EDB and IDB as a hypergraph $\mathbf{P_G}$. Each atom $a \in \mathbf{P}$ is represented as a vertex with a set of incoming hyperedges, where each hyperedge corresponds to one rule body of which a is the head. Also associated with each vertex is a set of outgoing hyperedge parts, each corresponding to one (positive or negative) appearance of a in the body of some rule $r \in \mathbf{P}$.

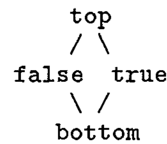
Computing the Current State

To determine the current state of the knowledge base, we compute the well-founded model [VRS91]. If the model is not total, we factor out the wf-residual – the unassigned vertex induced hypersubgraph [SS97] – and find the first assignment on the wf-residual that is a stable model by "guessing" and "checking" if such a model exists. [2]

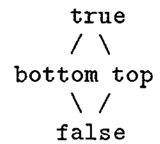
Van Gelder, Ross, and Schlipf present the original definition of the well-founded semantics in [VRS91]. Fitting used some of Ginsberg's work [Gin88] by incorporating the notion of a bi-lattice. He observes that the well-founded semantics uses the bi-lattice [Fit91]. Fitting also notes that in order to ascertain the truthfulness of a positive proposition, the well-founded semantics uses a **credibility** rating, which indicates how **provable** the proposition is. This is analogous to a progression up the truth ordering as defined in [Fit91]. To ascertain the truthfulness of negative propositions, however, the well-founded semantics uses an **evidence** rating, which indicates how much is known about the proposition. This is analogous to a progression up the knowledge ordering defined in the same work. The deduction algorithm of **INDED** uses Fitting's characterization of the well-founded semantics as a starting point. We cast his work into what we call the *Bilattice Algorithm*, which we show constructs the well-founded model of a normal logic program.

Definition 2.6 ($lub_{<_k}$ $lub_{<_t}$, $glb_{<_t}$) *We denote the least upper bound in the knowledge ordering over the set of truth values as $lub_{<_k}$. We denote the least upper bound in the truth ordering of the set of truth values as $lub_{<_t}$. We denote the greatest lower bound in the truth order of the truth values as $glb_{<_t}$. These partial orderings are defined in [Fit91] over the following lattices:*

KNOWLEDGE ORDERING



TRUTH ORDERING



The Bilattice Algorithm, Algorithm 2.7 below, utilizes all three operators: $glb_{<_t}$, $lub_{<_t}$, and $lub_{<_k}$ to ultimately assign each atom, or variable, a truth value. Each variable has two truth value fields: CREDIBILITY and EVIDENCE associated with the credibility and negative evidence assessments respectively. The outer loop updates the negative evidence (EVIDENCE) of each proposition, while the inner loop determines its credibility (CREDIBILITY). Each rule (head/body combination) has fields to track the combined truth valuation of the subgoals of the rule. The **neg_assessment** field contains the cumulative value of the negative subgoals, whereas the **pos_assessment** field contains the collective value of the positive subgoals. The **body_val** field represents the combination of pos_assessment and neg_assessment. Both the CREDIBILITY and the EVIDENCE of variable v are ultimately determined by operating on the body_vals of the rules for which v is the head. The final truth value assigned by this representation of the well-founded semantics for each variable v, is that variable's final EVIDENCE denoted EVIDENCE(v).

Algorithm 2.7 (Bilattice Algorithm) The algorithm is of quadratic complexity in the size of the program, where both the inner and outer loops are of linear complexity.

Input: hypergraph representation $\mathbf{P_G}$ of ground instantiation
Output: three-valued truth assignment $\mathcal{I}$

[2] If there is no stable model, we use the well-founded partial model as the background knowledge.

on vertices of $\mathbf{P_G}$

Initialize all variables
for each variable v
EVIDENCE(v) := bottom
CREDIBILITY(v) := false
Initialize all rules
for each rule r
body_val(r) := **true**
pos_assessment(r) := **true**
neg_assessment(r) := **true**

(OUTER LOOP)
Repeat until no change in
any variable's EVIDENCE
1. *perform all negative evidence assessments of all rules*
for each rule r with negative subgoals $\sim n_i$
neg_assessment(r) :=
$glb_{<t}(\neg(\mathbf{EVIDENCE}(n_i)))$
$0 \leq i \leq j$, where j is the number of negative subgoals in r

2. *determine new credibility*
for each variable v in $\mathbf{P_G}$
Initialize CREDIBILITY
for each variable v,
CREDIBILITY(v) := false
(INNER LOOP)
Repeat until no change
in any variable's CREDIBILITY
for each variable v in $\mathbf{P_G}$
for each rule r headed by v,
where r has positive subgoals p_i
pos_assessment(r) :=
$glb_{<t}(\mathbf{CREDIBILITY}(p_i))$
where $0 \leq i \leq m$,
m number positive subgoals in r
body_val(r) **:=**
$glb_{<t}$(**neg_assessment**(r),
pos_assessment(r))
Assign newly computed CREDIBILITY
CREDIBILITY(v) **:=**
$lub_{<t}(body_val_{i'})$
where $0 \leq i' \leq l$,
and l is num rules headed by v
(END INNER LOOP)

3. *determine new evidence*
for each variable v in the program,
$EVIDENCE_{k+1}(v)$:=
$lub_{<k}(CREDIBILITY_{k,n_k}(v),$
$EVIDENCE_k(v))$
(DEFINITE TERMINAL POINT)
(END OUTER LOOP)
(END OF ALGORITHM 2.7)

Theorem 2.8 *The Bilattice Algorithm computes the well-founded semantics.*

Proof Sketch:
Let $\mathcal{B_P}$ denote the output of the algorithm 2.7 where $\mathcal{B}_\mathcal{P}^\infty(\emptyset)$ denotes the least fixed point of $\mathcal{B_P}$, its final output, and $\mathcal{B}_\mathcal{P}^k(\emptyset)$ reflects the atom assignments after the kth iteration which comprise the sets $\langle trues\rangle_{\mathcal{B}_\mathcal{P}^k(\emptyset)}$, $\langle falses\rangle_{\mathcal{B}_\mathcal{P}^k(\emptyset)}$, and $\langle bottoms\rangle_{\mathcal{B}_\mathcal{P}^k(\emptyset)}$. In essence, $\mathcal{B}_\mathcal{P}^k(\emptyset)$ is equivalent to Van Gelder's alternating fixed point operator $\widetilde{A}$ defined in [Van93]. In particular,[3] $S_P(\widetilde{S_P}(\widetilde{I_k}))= \langle trues\rangle_{\mathcal{B}_\mathcal{P}^k(\emptyset)}$, $\widetilde{S_P}(\widetilde{I_k}) = \langle falses\rangle_{\mathcal{B}_\mathcal{P}^k(\emptyset)}$, and $\widetilde{I_k} \cap S_P(\widetilde{I_k})= \langle bottoms\rangle_{\mathcal{B}_\mathcal{P}^k(\emptyset)}$ for all k. In this paper, Van Gelder showed $\widetilde{A} \subseteq W_P$, the well-founded operator, and $\widetilde{A} \supseteq W_P$.
□

Upon leaving the deduction engine, the generated stable model remains fixed. It is provided to the induction engine as the background knowledge. After the induction of a hypothesis, the IDB is augmented with newly discovered intensional rules. A new ground instantiation is created and, hence, a new stable model is computed. A future area of research will investigate retaining previous stable models and building extensions on these. At this point in the research, however, each entry to the deduction engine creates a new stable model.

Induction Engine

We use a standard top-down hypothesis construction algorithm (learning algorithm) in **INDED**[LD94]. This algorithm uses two nested programming loops. The outer (covering) loop attempts to cover all positive examples, while the inner loop (specialization) attempts to exclude all negative examples. Termination is dictated by two user-input values to indicate sufficiency and necessity stopping criteria.

Algorithm 2.9 (Top-Down Construction)
This generic top-down algorithm has been assumed for the included ILP examples in this work and has been implemented in the system **INDED**. The algorithm is from [LD94] and forms the underpinnings of FOIL and GOLEM, among other top-down ILP systems.

Input: Target example sets $\mathcal{E} = \mathcal{E}^+ \cup \mathcal{E}^-$,
domain knowledge $\mathcal{B} = \mathcal{B}^+ \cup \mathcal{B}^-$,
SUFFICIENCY_STOP_CRITERION,
NECESSITY_STOP_CRITERION
Output: Intensional rule(s) of learned hypothesis $\mathcal{H}$

BEGIN ALGORITHM 2.9
while (Number Pos examples covered / $|\mathcal{E}^+|$ **)** $\leq$
SUFFICIENCY_STOP_CRITERION **) Do**
Make new intensional rule:
–head is target predicate of $\mathcal{E} = \mathcal{E}^+ \cup \mathcal{E}^-$
–body constructed as follows:
while (Num Neg examples covered / $|\mathcal{E}^-|$) $\geq$
NECESSITY_STOP_CRITERION) **Do**
- Append highest ranked literal
- Name variables of this chosen literal
end while

[3] see [Van93] for definition of notation

Add new intensional rule to
rule set $\mathcal{H}$ to be returned
end while
Return $\mathcal{H}$
END ALGORITHM 2.9

Negation as Failure-to-Learn

One commonly accepted form of deductive negation is called negation as failure-to-prove, and assumes propositions unable to be proved **true** to be **false** . Here, we present an inductive counterpart. Because data is often noise-laden or contradictory for other reasons, at times it is impossible to induce a targe predicate from given example sets.

Definition 3.10 (Measure of Inconsistency) *A value m is a measure of inconsistency if* $m = \frac{2*t}{|\mathcal{E}^+ \cup \mathcal{E}^-|}$ *where t is the number of tuples appearing in* $\mathcal{E}^+ \cap \mathcal{E}^-$.

Definition 3.11 (Negation as failure-to-learn) *A value c is a* coefficient of inconsistency *if c is a user input value defining the upper bound of measure of inconsistency m for a target predicate* $\mathcal{T}$. *If* $m > c$, *we say the negation of* $\mathcal{T}$ *is valid by* negation as failure-to-learn.

We implement this notion in **INDED** by giving the user the choice as to whether or not to induce the negation of the desired target predicate (as a negative intensional fact) when the measure of inconsistency exceeds the coefficient of inconsistency.

Examples of the proposed Framework

One of the challenges of this research is to identify classes of problems for which explicit negation and cycles in the background knowledge are helpful, if not necessary.

The Loser is Not the Winner

The following exemplifies the use of negation in the intensional rule set (IDB) of the background knowledge. This IDB expresses the notion of when a player is winning a game: that is, the last one who has a valid move. The IDB part of this example is borrowed from [VRS91]. The author extends it to learn the predicate *losing(X)*.

```
Input to deduction engine:
 IDB = { winning(X) <-- move(X,Y),~winning(Y). }
 EDB = { move(a,b) <-- .
         move(b,a) <-- . }
```

1. Compute the ground instantiation.

```
{ move(a,b) <-- .,
  move(b,a) <-- .,
  winning(a) <-- move(a,a), ~winning(a).,
  winning(a) <-- move(a,b), ~winning(b).,
  winning(b) <-- move(b,a), ~winning(a).,
  winning(b) <-- move(b,b), ~winning(b).   }
```

2. Compute a stable model, if there is one, or the well-founded otherwise.

```
{ move(a,b).,      move(b,a).,
  winning(a).,    ~move(a,a).,
 ~winning(b).,    ~move(b,b).    }
```

3. Using the stable model as $\mathcal{B} = \mathcal{B}^+ \cup \mathcal{B}^-$, and using other sets $\mathcal{E} = \mathcal{E}^+ \cup \mathcal{E}^-$, induce a new target predicate in the form of intensional rules.

```
Input to induction engine:
-------------------------
 E+ = { losing(b). }
 E- = {~losing(b). }

Learned Hypothesis:
------------------
     { losing(X) <-- ~winning(X). }
```

4. Send the intensional rule back to deduction engine to create a new ground instantiation.

```
{  move(a,b) <-- .,
   move(b,a) <-- .,
   winning(a) <-- move(a,a), ~winning(a).,
   winning(a) <-- move(a,b), ~winning(b).,
   winning(b) <-- move(b,a), ~winning(a).,
   winning(b) <-- move(b,b), ~winning(b).
   losing(a)  <-- ~winning(a).,
   losing(b)  <-- ~winning(b).                  }
```

Learning Negatively Recursive Rules

Programs that do not have any stable models possess odd negative cycles of dependencies [YY90]. These are sets of rules forming dependency chains with an odd number of negations such as in the rule $p \leftarrow \neg p$ which forms an odd negative cycle of length one. Knowledge of the presence of such relationships in the ground instantiation can shed light on the consistency of the current knowledge base. We show here how we are able to learn $p \leftarrow \neg p$ using **INDED** and then offer a real-world scenario observing that differing sources of data from varying contexts can produce such rules.

Learning $p \leftarrow \neg p$

```
Input to deduction engine:
 IDB = { }
 EDB = {p(c) <--.,
        p(d) <--.,
        p(a) <-- p(b).,
        p(b) <-- p(a).}
```

1. Compute the ground instantiation.

```
{ p(c) <--.,
  p(d) <--.,
  p(a) <-- p(b).,
  p(b) <-- p(a).}
```

2. Compute a stable model which in this case is the well-founded model.

```
{ p(c).,
  p(d).,
 ~p(a).,
 ~p(b).}
```

3. Using the stable model as $\mathcal{B} = \mathcal{B}^+ \cup \mathcal{B}^-$, and using other sets $\mathcal{E} = \mathcal{E}^+ \cup \mathcal{E}^-$, induce a new target predicate in the form of an intensional rule.

```
Input to induction engine
-------------------------
 E+ = {  p(a).,  p(b).}
 E- = { ~p(c)., ~p(d).}

Learned Hypothesis
------------------
 { p(X) <-- ~p(X).}
```

By learning a target predicate already appearing in the stable model of the original ground instantiation (in this case "p"), and by using the constants of **false** atoms of the stable model in the positive example set of this target, we are able to learn a negatively recursive rule. This sequence of events can happen in real life when we acquire data from two disparate sources. We illustrate this in the following example where anamnestic data (data acquired from a human survey) used by the marketing department of a hypothetical company, is used to relearn a predicate in the current knowledge base which was determined by experimental data acquired from the company's R&D organization.

Which soap is best?

A customer evaluation survey tells us soap A removes more dirt than soap B. We represent the customer data below as $\mathcal{E} = \mathcal{E}^+ \cup \mathcal{E}^-$. The lab tells us that soap B has more surfactants (cleaning agents) than soap A. We represent the laboratory data below as $\mathcal{B} = \mathcal{B}^+ \cup \mathcal{B}^-$.

```
Input to deduction engine
-------------------------
EDB: { max_surfactant(soap_B).,
      ~max_surfactant(soap_A)}
IDB: {superior_soap(X) <-- max_surfactant(X).}
```

1. Compute a stable model which in this case is the well-founded model.

```
{  max_amount_surfactant(soap_B).,
   superior_soap(soap_B).,
  ~max_amount_surfactant(soap_A).,
  ~superior_soap(soap_A).    }
```

2. Using the stable model as $\mathcal{B} = \mathcal{B}^+ \cup \mathcal{B}^-$, and using other sets $\mathcal{E} = \mathcal{E}^+ \cup \mathcal{E}^-$, induce a new target predicate in the form of an intensional rule.

```
Input to induction engine
-------------------------
E+ = { superior_soap(soap_A).}
E- = {~superior_soap(soap_B).}

Learned Hypothesis
------------------
{superior_soap(X) <-- ~superior_soap(X).}
```

Upon detection of such a contradiction, a reasonable question might be: *"Why do most customers perceive soap A to be more effective? Are there other characteristics or attributes that the customers are detecting?"* An examination of the anamnestic data might uncover a pattern that reveals some other attribute of the soap, such as fragrance or color, to be causing the respondents to state that soap A is the superior soap.

Status of Current Research

We are currently studying and applying this work to two problems. The first uses **INDED** in a data mining context to extract meta-patterns from data [SJM99]. Because of the internal hypergraph representation of the deduction engine, detection of cycles of dependencies in data is performed quickly and accurately. Our future work in this vein, consists of identifying near and partial cycles.

Secondly, we are studying the diagnosis of Lyme disease which is quite difficult because the disease mimics many other diseases [Bur97] [Sch89]. In this work, we will use the inductive and deductive capabilities of **INDED** to generate rules relating to the diagnosis of Lyme disease. Because the symptom sets of varying patients seem to be laced with contradictions, we will capitalize on the ability of the well-founded semantics to handle contradictions. We will also use the ability of **INDED** to handle contradictions by generating rule sets which form odd negative cycles of dependencies. We expect these odd cycles to provide a new diagnosis approach, or at least accurately codify current diagnosis scenarios. We are currently obtaining data and casting it into a form recognizable by **INDED**.

Conclusion

This work applied stable and well-founded models to the background knowledge of inductive logic programming. We have presented an implementation which performs nonmonotonic deduction using stable and well-founded models, and iterative top-down induction, called **INDED**. System **INDED** has the ability to learn positively and negatively recursive rules, as well as to learn negated target predicates under the justification of negation as failure-to-learn. By juxtaposing contexts, **INDED** also provides a source of anomalous logic programs by inducing rules that form negatively recursive chains of dependencies, thus presenting a real-world source of such rules. Although these rules are not necessarily desirable, the existence of such rules in a consistent knowledge base has long been questioned. This work has shown that through the juxtaposition of contexts represented by sets of logic facts, such rules can be induced in a discovery environment.

References

[Bur97] Joseph J. Burrascano, Jr., M.D. Managing lyme disease diagnostic hints and treatment guidelines for lyme borreliosis. The Lyme Disease Network of NJ, Inc., http://www.lymenet.org.

[Doy79] J. Doyle. A truth maintenance system. *Artificial Intelligence*, 12:231–272, 1979.

[Fit91] Melvin C. Fitting. Well-founded semantics generalized. *Annals of Mathematics and Artificial Intelligence*, pages 71-84, 1991.

[GC96] C. Giraud-Carrier. Flare: Induction with prior knowledge. *Proceedings of Expert Systems'96*, 1996.

[GCM95] C. Giraud-Carrier and T. Martinez. An integrated framework for learning and reasoning. *Journal of Artificial Intelligence Research*, 3:147–185, 1995.

[GL90] Michael Gelfond and Vladimir Lifschitz. The stable model semantics for logic programming. In *Proceedings of the Fifth Logic Programming Symposium*, pages 1070–1080, 1990.

[Gin88] M.L.Ginsberg. Multivalued logics: a uniform approach to reasoning in artificial intelligence. In *Computational Intelligence*, 4, 1988.

[LD94] Nada Lavrac and Saso Dzeroski. *Inductive Logic Programming.* Ellis Horwood, Inc., 1994.

[MT92] A. Marek and M. Truszczynski. Revision specifications by means of programs. Technical report, University of Kentucky, University of Kentucky, Lexington, KY, 1992.

[Mug92] Stephen Muggleton, editor. *Inductive Logic Programming.* Academic Press, Inc, 1992.

[Sch89] Mary A. Schilling. Occurrence of the spirochaete, borrelia burgdorferi, and the rickettsiae, rickettsia rickettsii, in eastern central Missouri. Master's thesis, Saint Louis University, 1989.

[Sei97] Jennifer Seitzer. Stable-ILP: Exploring the added expressivity of negation in the background knowledge. In *Proceedings of the Frontiers in Inductive Logic Programming Workshop.* Fifteenth International Joint Conference on Artificial Intelligence, Nagoya, Japan, 1997.

[SJM99] J. Seitzer, J.P.Buckley, and A. Monge. Metapattern extraction: Mining cycles. *Proceedings of the Twelfth International FLAIRS Conference*, 1999. To appear.

[SS97] Jennifer Seitzer and John Schlipf. Affordable classes of normal logic programs. In *Lecture Notes in Artificial Intelligence: Logic Programming and Nonmonotonic Reasoning Fourth International Conference Proceedings*, 1997.

[Van93] Allen VanGelder. The alternating fixpoint of logic programs with negation. *Journal of Computer and System Sciences*, 47:185–221, 1993.

[VRS91] A. VanGelder, K. Ross, and J. Schlipf. The well-founded semantics for general logic programs. *Journal of the ACM*, 38(3):620–650, July 1991.

[YY90] Jia-Huai You and Li Yan Yuan. Three-valued formalization of logic programming: Is it needed? In *Proceedings of the Ninth ACM SIGACT-SIGMOD -SIGART Symposium on Principles of Database Systems*, pages 172–182, 1990.