1. We are going to set up spatial (i.e. "space-ial") interpretations for Logic Programs (i.e. "pure" Prolog programs.) This will be our approach to the semantics of Logic Programming. Applications in the language Prolog will serve from time to time as examples for what we explore in the course, and the basic procedural aspects behind how the language works are important to know about. In recent years a variation of logic programming, called answer-set programming, has become important to researchers and developers in the intersection of Logic Programming and AI. Poke around the web to get a feel for what people are doing in these areas: answer-set programming and, more generally, logic programming.

2. To get anywhere with a logic-based approach to reasoning and knowledge, even to understand the limitations of such an approach with any precision, one must first understand the basics of logic, particularly first-order logic - which can be seen as the "assembler language" for all higher-type and higher-level logics. It has been said - I have not been able to find the proper attribution - that first-order logic is the assembler language of God, or the Universe, or whatever really, really big idea you have in mind. In the interest of full disclosure: I am sympathetic to that point of view. Look up the syntax (i.e. grammar) of propositional logic. See, for example, the entry on propositional logic in the Wikipedia.

3. The connective $\leftarrow$, as in $\psi \leftarrow \varphi$ is the converse of $\rightarrow$. In other words $\psi \leftarrow \varphi$ can be taken as a macro for $\varphi \rightarrow \psi$. We bother with this connective because of the syntax of Prolog. Every formula of propositional logic is equivalent to a conjunction of formulas, each of which has the following form:

$$A \leftarrow L_1, \ldots, L_n$$

for some $n \geq 0$, where $A$ is an atom, or the truth constant $\text{false}$, and each $L_1$ is a literal. An atom is a literal, and the negation of an atom is a literal. Nothing
else is a literal. For example:

\[
\neg(p \lor \neg q) \leftrightarrow r \\
\equiv (\neg(p \lor \neg q) \rightarrow r) \land (r \rightarrow \neg(p \lor \neg q)) \\
\equiv ((\neg p \land q) \rightarrow r) \land (r \rightarrow (\neg p \land q)) \\
\equiv (r \leftarrow \neg p \land q) \land (r \rightarrow \neg p) \land (r \rightarrow q) \\
\equiv (r \leftarrow (\neg p \land q)) \land (\text{false} \leftarrow r \land p) \land (q \leftarrow r) \\
\equiv (r \leftarrow (\neg p \land q)) \land (\neg r \land p) \land (q \leftarrow r)
\]

The last line above is explained by the convention that we just leave out the truth constant false on the left side of \leftarrow-formulas when it would otherwise occur. That convention accords with Prolog’s syntax for queries. Moreover, top-level conjuncts in

\[
(r \leftarrow (\neg p \land q)) \land (\text{false} \leftarrow r \land p) \land (q \leftarrow r)
\]

are just listed as:

\[
\begin{align*}
& r \leftarrow \neg p, q \\
& q \leftarrow r \\
& \leftarrow r, p
\end{align*}
\]

Conjunction symbols on the right side of the \leftarrow are abbreviated to commas. The first two formulas in the above list are the normal program clauses of a logic program that consists of the set of these two clauses. The third clause is called a negative clause, or a query. In a logic programming context we call a normal program clause just a clause, or sometimes a normal clause. A Horn clause is a normal clause without negations on the right-hand side. The atom on the left-hand side of a normal clause is called the head of the clause. The right-hand side of the clause is called the body of the clause. So normal clause bodies are conjunctions of literals. Do: Find a logic program together with necessary queries that is equivalent to the formula

\[
\neg(p \rightarrow q) \leftrightarrow p \lor r
\]

One last point about the propositional form of logic programs: a program clause

\[
A \leftarrow L_1, \ldots, L_n
\]

is logically equivalent to the query

\[
\leftarrow \neg A, L_1, \ldots, L_n
\]
4. Logic programs go beyond propositional logic. **Do:** Look up the syntax, or a syntax - different authors use different syntactical conventions, for first-order logic (FOL). The Wikipedia is again a pretty good source.

5. Each FOL formula has a propositional form: the propositional logic formula you get by regarding each quantified formula $\forall x \varphi$ or $\exists x \varphi$ as if it were an atom. For example, $\exists x \forall y (p(f(a,x),y) \land q(y)) \rightarrow q(b)$ has the same form as $r \rightarrow q$. The whole formula $\exists x \forall y (p(f(a,x),y) \land q(y))$ is regarded as if were just an atom. That being the case, we could present a logic program with queries as an equivalent form of a FOL formula, but this time clause heads and the literals in clause bodies might be complicated FOL formulas. It doesn’t seem like that wins us much. But, imagine a clause that looks like

$$A \leftarrow \ldots, \exists x \varphi, \ldots$$

By using prenex operations to factor out the $\exists$ quantifier, we get

$$\forall x [A \leftarrow \ldots, \varphi, \ldots]$$

(We might have to change bound variables - and you might need to look up prenex forms.) Universal formulas embedded in clause bodies are more complicated. If we had

$$r(y) \leftarrow \ldots, \forall x \varphi(x,y), \ldots$$

we replace it with

$$r(y) \leftarrow \ldots, \neg \exists x \neg \varphi(x,y), \ldots$$

and replace that with

$$r(y) \leftarrow \ldots, \neg p(y), \ldots$$

$$p(y) \leftarrow \exists x \neg \varphi(x,y)$$

and then factor out the $\exists x$. $p$ is a new predicate symbol not occurring elsewhere in the formulas under consideration. It turns out that the resulting set of clauses is not equivalent to what we started with, but it is a conservative extension in the supported models semantics of logic programs. **Do:** Begin with

$$p(x) \leftarrow \forall y \forall z q(x,y,z)$$

and eliminate the embedded universal quantifiers. (The variable $x$ is implicitly universally quantified on the outside of the clause.)
1. Next, we have to turn to topics such as supported models. In the basic version of the semantics we need two concepts: Herbrand interpretations and the one-step consequence operator. Obtaining other versions will then be fairly simple. A language for first-order logic is determined by picking a set of constant symbols, function symbols, and predicate symbols - and assigning an arity to each function and predicate symbol chosen - this chosen set of symbols is called the set of nonlogical symbols of the language. An arity is a nonnegative integer. The arity of a symbol is the number of arguments it takes. (Constant symbols are function symbols that have arity 0.) A simple logic program $P$ is given below:

$$
\text{append}(\text{nil}, L, L) \leftarrow \\
\text{append}(\text{cons}(H, L), M, \text{cons}(H, N)) \leftarrow \text{append}(L, M, N)
$$

**Do:** Identify the predicate, function, and constant symbols in the above program, and indicate the arity of each symbol. That collection of symbols with their arities determines the smallest language in which the clauses of the program can be formulas. That language is called the language of $P$, and this notion of the language of a set of formulas applies to any set of formulas, not just the clauses in the above logic program.

2. The *Herbrand universe* of a language is the set of all terms in the language that do not have variables occurring in them. Such a term without variables is called a *ground* term. **Do:** Describe the Herbrand universe of the language of the program $P$ from the previous item.

3. Note how the phrase structure tree (look up such terms as necessary) of a term is actually a vertex-labeled tree. The height of a term is the height of its phrase structure tree. (*Height* is sometimes called *depth.*) **Do:** List all terms in the Herbrand universe of the language of $P$ that have height at most 3. You might want to do this task with a program.

4. **Do:** Install a Prolog on your personal computer. Run the following program to yield all solutions:

$$
\text{height}(\text{nil}, 0).
\text{height}(\text{cons}(X,Y), N) :- \text{successor}(N2, N), \text{leq}(N1, N2), \text{height}(X, N1), \text{height}(Y, N2).
\text{height}(\text{cons}(X,Y), N) :- \text{successor}(N1, N), \text{less}(N2, N1), \text{height}(X, N1), \text{height}(Y, N2).
$$
leq(0,N).
leq(M,N) :- successor(M1,M), successor(N1,N), leq(M1,N1).

less(M,N) :- successor(M,M1), leq(M1,N).

successor(0,1).
successor(1,2).
successor(2,3).

heightAtMost(X,N1,N) :- leq(N1,N), height(X,N1).

5. Do: Explore the execution behavior of the program by reordering the clauses and the atoms within the clause bodies. Beware of runaway recursions.

6. The Herbrand base of a language is the set of all atomic formulas of the language that do not have variables occurring in them. Such formulas are called ground atomic formulas, or synonymously, just ground atoms. Do: Describe the Herbrand base of the language of P.

7. Do: List all ground atoms in the Herbrand base of the language of P of height at most 3.

8. An Herbrand interpretation of a language is a subset of the Herbrand base of the language. Do: Suppose we have a language with exactly two 2-ary predicate symbols, p and q, and exactly 2 constant symbols, a and b. Determine how many Herbrand interpretations of this language there are.

Material for: February 13, 2006

1. Do: Use GNU-prolog to explore

   q(a) :- p(Y,f(Y)).
   q(b).

   p(X,X).

   and
p(a0).
p(a) :- s(X), !, r(Y).
p(b).

s(c1).
s(c2).

r(d1).
r(d2).

and

p(a0).
p(a) :- s(X), r(Y).
p(b).

s(c1).
s(c2).

r(d1).
r(d2).

Material for: February 15, 2006

1. Recursion. Example: The Towers of Hanoi: The setup is that there are three towers (poles, really) left, middle, right. There are 64 heavy stone slabs with holes in their middles, each slab of a different diameter. Initially, the 64 slabs are placed on one of the towers, for example, left, in such a way that no larger slab occurs anywhere above a smaller slab on the tower. The task is move the slabs from the left tower to the middle tower. There are three rules: (1) no larger slab is allowed to occur anywhere above a smaller slab on the same tower at any time; (2) only one slab at a time is allowed to be moved; (3) slabs can only be moved among the towers - you can’t put a slab down on the ground, for example. Allegedly there was a mystical cult that used be located near Hanoi, Vietnam that believed that the purpose of the universe is to accomplish this task, after which, the universe will cease to exist. They busily worked away at it. (Assuming that on average one slab per 10 seconds gets moved, calculate when the universe will end. cf. “The Nine Billion Names of God” by Arthur C. Clarke.) Here is a prolog program that writes out the sequence of moves.
move(1,From,To,_) :-
    write('Move top disk from '),
    write(From),
    write(' to '),
    write(To),
    nl.

move(N,From,To,Spare) :-
    N>1,
    M is N-1,
    move(M,From,Spare,To),
    move(1,From,To,_),
    move(M,Spare,To,From).

Do: Explain why it works. Look up how numbers, and is works in Prolog.

2. Here is a procedural if-then-else in Prolog

\[
p :- q, !, r.
p :- s.
\]

The idea is that the call of p succeeds by first calling q so that if the call to q succeeds, then for p to succeed, r must succeed, and of the call to q fails, then p gets to try to succeed by calling s. I.e.

\[
p :- (if \ q \ then \ r \ else \ s).
\]

Do: Make up and test some examples of the if-then-else construction as given here.

3. What about negation? Specifically, what about negation in clause bodies, as in:

\[
p(X) :- \text{not} \ q(X).
\]

\[
q(a).
\]

One approach to implement negation in Prolog is through the use of the \text{cut-fail} combination. Consider
p(X) : - not_q(X), r(X).
p(c).

not_q(X) :- q(X), !, fail.
not_q(X).

q(a).

r(a).
r(b).

**Do:** Replace \( p(X) : - \text{not}_q(X), r(X) \). by \( p(X) :- r(X), \text{not}_q(X) \). in
the above program and explore the backtracking behavior of the resulting pro-
gram.

4. The cut-fail approach is partially built in to Prolog using the syntax `\+` as in

\[
p(X) :- \text{\Small plus} q(X), r(X).
p(c).
q(a).
r(a).
r(b).
\]

Read the symbol `\+` casually as ‘not’ but do not forget that it has a special,
and logically unsound (without further restrictions) semantics. **Do:** Make up
an example that shows that backtracking can proceed across `\+`

5. **Explore:**

\[
\text{not}_q(X) :- q(X), !, \text{fail}.
\text{not}_q(X).
q(a).
\]

In particular, try the queries `not_q(a)` and `not_q(b)`.

6. The cut-fail combination reflects the idea of *negation by failure*, or as it is
sometimes called, *negation as failure*. This is the idea that if an assertion \( A \)
cannot be proved from what is given, then \( A \) must be false. Without restrictions
this is an unsound notion. There are easy counterexamples. Suppose that we
are given that everyone in a certain situation is either male or female and that in that same situation, Leslie is 21. We cannot prove from what we are given that (M) Leslie is a male, and we cannot prove that (F) Leslie is a female. Since we can’t prove (F), then by negation as failure, Leslie is not a female. Similarly, Leslie is not a male. Therefore, since Leslie is not a male, and in the situation we are considering, Leslie is either male or female, Leslie is a female. Contradiction! By negation as failure, it similarly follows that Leslie is a male. Again a contradiction.

But wait. Didn’t we just prove using negation as failure that Leslie is a female and that Leslie is a male? So negation as failure won’t allow us to conclude that Leslie is not a male and that Leslie is not a female, and so we can no longer conclude that Leslie is a female and that Leslie is a male. But wait, since we can’t after all prove that Leslie is a female, then by negation as failure, Leslie is a male ... . And around we go..

There seems to be something unstable about this. The idea that if we cannot prove \( A \), then \( A \) must be false, is the idea that we are given complete information about a situation. Case closed, so to speak. This idea is called the closed world assumption (CWA).

Notice that if we really are given a complete, exhaustive, listing of all the facts of a situation, then reasoning using the closed world assumption doesn’t lead to contradictions. But if we had such a list, we wouldn’t need to do much reasoning. One of the objectives, however, is to find compact minimalist approaches to storing data in order to minimize the need for memory resource and minimize searching time, so as to trade off the need for memory to high-powered deductive techniques. Another objective, is to be able to compute what the logical possibilities are for how the given information could be extended so that using the CWA would be valid.

In the world of automated reasoning and knowledge representation there are a number of responses to this challenge that have arisen in the area known as nonmonotonic reasoning (NMR).

Among the responses to this challenge in the logic programming world specifically, is the idea of stable models, and based on that, the idea of answer-set programming. The situation involving Leslie could be represented like this:

\[
\text{male}(X) :\neg \text{female}(X). \\
\text{female}(X) :\neg \text{male}(X). \\
\text{age21}(\text{leslie}).
\]
**Do:** Find all of the Herbrand models of the above program. Two Herbrand models correspond to how Leslie’s situation could be. One of the Herbrand models does not account for the either-or categorization of Leslie’s gender as being exclusive. Which model?

**Material for: February 20, 2006**

Many of the following items are adapted from the research report *Set-based Logic Programming* by Howard A. Blair (Dept. of Electrical Engineering and Computer Science, Syracuse University), Victor W. Marek (Dept. of Computer Science, University of Kentucky) and Jeffrey B. Remmel (Department of Mathematics, UC San Diego).

1. There are miscellaneous Prolog program examples through the link Prolog Examples. **Do:** Run the test1 example from the example file mcsam.pl. It will probably fail unless you are using a Prolog with the built-in predicate `writeseqnl/1`. You will find that predicate in the example file `utilities.pl`. There are lots of built-in predicates in GNU Prolog that are not other Prologs and vice-versa. There is a Prolog standard, but it is not stringently followed.

2. Our next goal is to understand the basic set-up of answer-set programming. For any given set $J \subseteq HB$, where $HB$ is the Herbrand base of a first order language under consideration, we define the Gelfond-Lifschitz transform [cf. GELFOND, M. AND LIFSCHITZ, V. “The stable model semantics for logic programming”, in *Proc. of the International Joint Conference and Symposium on Logic Programming*, MIT Press, 1070–1080, 1988] of a program $P$, $GL_J(P)$, in two steps. First we consider all ground instances $C$ of clauses in $P$. If $J \models L_i$ for some negative literal $L_i$ in the body of $C$, then the we eliminate clause $C$. If not, then we replace $C$ by the Horn clause obtained from $C$ by removing all negative literals from the body of $C$. The $GL_J(P)$ consists of $EBP(P)$ plus the sets of all Horn clauses produced by this two step process. Thus $GL_J(P)$ is a Horn program so that the least fixed point of $T_{GL_J(P)}$ is defined. Then we say that $J$ is a *stable model* of $P$ if and only if $J$ equals the least model of $GL_J(P)$. For example, if $P$ consists of only one clause

$$p \leftarrow q, \neg p$$

There are four Herbrand interpretations of $P$, $\emptyset$, $\{p\}$, $\{q\}$, and $\{p,q\}$. The Gelfond-Lifschitz transform of $P$ with respect to each of these four Herbrand interpretations is

$$
GL_{\emptyset}(P) = p \leftarrow q \\
GL_{\{p\}}(P) = \emptyset \\
GL_{\{q\}}(P) = p \leftarrow q \\
GL_{\{p,q\}}(P) = \emptyset
$$
(Remember that a logic program is a set of clauses. So, $\emptyset$ is a logic program.) The least fixed points of the two resulting logic programs are given by

$$\text{lfp}(p \leftarrow q) = \emptyset$$
$$\text{lfp}(\emptyset) = \emptyset$$

So, the only stable model of $P$ in this example is the empty interpretation; i.e. both $p$ and $q$ are false. Notice that any stable model of a logic program $P$ is a supported model of $P$. **Prove it.**

3. **Do:** Find all of the stable models of

$$p \leftarrow q, \overline{q}, \neg p$$
$$q \leftarrow \neg \overline{q}$$
$$\overline{q} \leftarrow \neg q$$

We temporarily call this program $\mathcal{A}$. $\mathcal{A}$ contains two basic tricks of default reasoning and answer-set programming: (1) The use of renaming to introduce a protected form of negation to “fool” defaults. $\overline{q}$ “looks like” a positive literal in the above clauses from the point of view of the Gelfond-Lifschitz transform, but we think of it as another way of expressing $\neg q$. (2) The first clause is an instance of the general form:

$$\text{dummy} \leftarrow \varphi, \neg \text{dummy}$$

4. In any stable model of any program that only uses the dummy predicate only in the way used in the previous clause, any ground instance of the formula $\varphi$ must be false. **Prove it.** In Smodels, the answer-set programming system we will use below, a clause such as the one above is simply written as

$$\varphi$$

and is called an *integrity constraint* in the context of answer-set programming, as opposed to being called a *query*, as it would be in a Prolog context.

5. To use the Smoodels system, one first prepares an answer-set program. There are examples of answer-set programs through the link Smoodels Examples. For example, the file color1.lp (note the filename extension is *lp*, not *pl* as is typically the case for Prolog examples) is available through the preceding link, along with the Lparse 1.0 Users Manual, which is really the Smoodels manual (and is a .pdf file converted from a Postscript file.) What were they thinking? Section 1.3 of the Lparse manual is called “A Practical Example”, and discusses the contents of the file color1.lp. **Run:** color1.lp and stab1.lp. **Explore:** The other examples. To run a file named name, one enters
from the command line. You will find the lparse and smodels models in the directory /home/cisfac/blair/Smodels/bin/bin on the college’s Unix system.

6. Have a look through the link Prolog Code Example for some serious Prolog code (just to get a basic impression of how the language is used to code realistic software.)

Material for: February 22, 2006

1. Assignment to be submitted Deadline: February 27, 2006 by 2:15pm. Modify the hanoi towers program so that it prints out a sequence of tower configurations horizontally, as in

```
<p>| | |
| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>----</td>
<td>----</td>
</tr>
</tbody>
</table>
| -----|----- | --|--
```

```
<p>| | |
| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>----</td>
<td>----</td>
</tr>
</tbody>
</table>
---|---
```

```
<p>| | |
| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>----</td>
<td>----</td>
</tr>
</tbody>
</table>
---|---
---|---
```

```
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>--</td>
<td>--</td>
</tr>
<tr>
<td>---</td>
<td>---</td>
</tr>
<tr>
<td>----</td>
<td>----</td>
</tr>
<tr>
<td>-----</td>
<td>-----</td>
</tr>
</tbody>
</table>
```

```
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>--</td>
<td>--</td>
</tr>
<tr>
<td>---</td>
<td>---</td>
</tr>
<tr>
<td>----</td>
<td>----</td>
</tr>
<tr>
<td>-----</td>
<td>-----</td>
</tr>
</tbody>
</table>
```

```
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>----</td>
<td>----</td>
</tr>
<tr>
<td>-----</td>
<td>-----</td>
</tr>
</tbody>
</table>
```

```
2. Three-valued logic: Suppose that \( I \) assigns truth-values from the set \( \{ t, f, \bot \} \) to each ground atom, in the Herbrand base of a particular language for first-order logic, and also assigns one of these truth-values to each negation of each ground atom. We can think of \( \bot \) as the “undefined value”, as we might if we had a Boolean variable was declared but was not yet assigned. We could extend the truth-value assignment to all formulas in several possible ways. The following is one of these ways, intended to correspond to the idea that if any part or instance of a formula is unassigned, then so is the whole formula.

\[
\alpha_I(A \land B) = \begin{cases} 
  t & \text{if } \alpha_I(A) = \alpha_I(B) = t \\
  f & \text{if } \alpha_I(A) = f \text{ and } \alpha_I(B) \neq \bot \\
  f & \text{if } \alpha_I(B) = f \text{ and } \alpha_I(A) \neq \bot \\
  \bot & \text{otherwise.}
\end{cases}
\]

Similarly,

\[
\alpha_I(A \lor B) = \begin{cases} 
  f & \text{if } \alpha_I(A) = \alpha_I(B) = f \\
  t & \text{if } \alpha_I(A) = t \text{ and } \alpha_I(B) \neq \bot \\
  t & \text{if } \alpha_I(B) = t \text{ and } \alpha_I(A) \neq \bot \\
  \bot & \text{otherwise.}
\end{cases}
\]

For the three-valued conditional:

\[
\alpha_I(A \rightarrow B) = \begin{cases} 
  t & \text{if } \alpha_I(A) = \alpha_I(B) = t \\
  t & \text{if } \alpha_I(A) = f \text{ and } \alpha_I(B) \neq \bot \\
  f & \text{if } \alpha_I(A) = t \text{ and } \alpha_I(B) = f \\
  \bot & \text{otherwise.}
\end{cases}
\]

The quantifiers are interpreted schematically.

\[
\alpha_I(\forall x \varphi(x)) = \begin{cases} 
  t & \text{if } \alpha_I(\varphi(e)) = t, \text{ for all constants } e \\
  f & \text{if } \alpha_I(\varphi(e)) = f, \text{ for some constant } e \\
  \bot & \text{and } \alpha_I(\varphi(e')) \neq \bot, \text{ for all constants } e' \\
  \bot & \text{otherwise.}
\end{cases}
\]

\[
\alpha_I(\exists x \varphi(x)) = \begin{cases} 
  t & \text{if } \alpha_I(\varphi(e)) = t, \text{ for some constant } e \\
  \bot & \text{and } \alpha_I(\varphi(e')) \neq \bot, \text{ for all constants } e' \\
  f & \text{if } \alpha_I(\varphi(e)) = f, \text{ for all constants } e \\
  \bot & \text{otherwise.}
\end{cases}
\]

Finally, we define \( I \models \varphi \) if and only if \( \alpha_I(\varphi) = t \).

3. What about negations of formulas other than atoms? To assign a truth-value to a formula \( \neg \varphi \), first (recursively) obtain the truth-value \( \alpha_I(\varphi) \) of \( \varphi \). Next apply \( \alpha_I \) to \( \varphi \rightarrow \text{false} \), where \( \text{false} \) is just some atom that has been assigned \( f \) by \( \alpha_I \).
4. There is a potential incoherence in the previous item: The approach to negation will agree with how we assign truth values to negative literals if, and only if, we arrange for $\alpha_I(\neg A) = \alpha_I(A \rightarrow \text{false})$.

5. Notice that the preceding definitions allow us to assign truth-values to all formulas once we know how to assign truth-values to literals. These ideas can be extended to continuous truth-values, and the schematic approach to quantifiers allows us to greatly expand our ideas about how to interpret formulas of logic. Continuous truth-values can be used to represent evidence weights for or against a statement, or degrees of knowledge about a statement.

6. Define $A \mid B$ to be an abbreviation of $\neg(A \land B)$. What is $\alpha_I(A \mid B)$ in terms of $\alpha_I(A)$ and $\alpha_I(B)$? What is the relationship between $\alpha_I(A \rightarrow B)$ and $\alpha_I(\neg A \lor B)$, for any formulas $A$ and $B$? Do the Demorgan Laws hold?

7. We will apply the ideas in the last few preceding items later on. First, we need to develop some intuition for some finer points of logic and how logic connects to some other mathematically useful ideas.

8. Consider the set of all propositional formulas $\mathcal{F}_{\{p,q\}}$ generated by two Boolean variables $p$ and $q$. Show that $\mathcal{F}_{\{p,q\}}$ is infinite.

9. Let $\sim$ be the relation defined on $\mathcal{F}_{\{p,q\}}$ by: $A \sim B$ iff $A \equiv B$ is a tautology. Show that $\sim$ is an equivalence relation.

10. Show that $\sim$ is a congruence with respect to $\neg$ and $\lor$. For example, show that: if $A \sim B$, then $\neg A \sim \neg B$. This makes $\sim$ a congruence with respect to all Boolean connectives. Why?

11. Define the relation $\sqsubseteq$ on $\mathcal{F}_{\{p,q\}}$ by: $A \sqsubseteq B$ iff $A \rightarrow B$ is a tautology. ($\sqsubseteq$ is read (counterintuitively) as “is at least as strong as” - allowing for the possibility of a formulas being stronger than themselves.) Show that $\sqsubseteq$ is a partial order on $\mathcal{F}_{\{p,q\}}$.

12. Show that $\sim$ is a congruence with respect to $\sqsubseteq$. After finishing the tasks in the items up to this point you will have shown that the Boolean connectives and the “at least as strong as” partial order lifts to the equivalence classes determined by $\sim$.

13. Verify that the following diagram depicts the “is at least as strong as” partial ordering on the equivalence classes determined by $\sim$. The resulting system, or algebraic structure $(\mathcal{F}_{\{p,q\}}, \sim)$ is an example of what is called a quotient.
The Hasse diagrams above depict the projection of a 4-cube into 2-space, much as we can draw a 3-cube on a flat piece of paper. Another way to depict a 3-cube in 2-space is cut it on certain edges and unfold it, flattening out the result. Here is a 2-space projection of a 4-cube cut and unfolded in 3-space.
1. We will look at several miscellaneous topics that will give us the beginning of a collection of tactics for handling various knowledge representation and reasoning tasks in a reasonably efficient way. They techniques are heuristics; they are not guaranteed to always deliver efficiency, but they are still useful. **Do:** Read sections 3.7 and 3.8 in *Brachman and Levesque*. 

(Material for: February 27, 2006)
2. The first topic is a type of reification. To reify (“re-i-fie”) is to regard something abstract as if it had concrete existence. (A paper by Willard Van Orman Quine, “Events and Reification”, is available on-line. Search ‘Quine’ and ‘Reification’ at http://scholar.google.com
This paper isn’t required, but if you are intellectually hungry, you should look it over. It is directly relevant to the current topic, and Quine is high-powered.) One might argue that everything that each of us takes as real is the result of reification. Be that as it may, the tactic is useful.

3. Consider the proposition

Carlos is the chairperson

We might represent this proposition in FOL (recall that FOL means first-order logic) as

chairperson(carlos)

This representation leaves a huge number of questions. For example here are just the first few we might think of.

- Carlos who?
- Of what?
- When? Start time? Stop time? Did Carlos ever stop being chairperson and did he start again as chairperson later on?
- Is there more than one chairperson?
- Did whatever Carlos is chairperson of become something else? (This item might seem like extreme nit-picking, but it is actually true of the EECS department). Did Carlos stay as chairperson, of did he have different administrative positions before and after the transition?
- Is Carlos the assemblage of all of Carlos’s parts?
- Almost all of the matter of which Carlos’s body is made was not part of Carlos’s body when he became chairperson. Just who is this chairperson?

To represent factual answers to the first two or three questions might try the following alterantives:

chairperson(carlos, hartmann)
chairperson(carlos, hartmann, eecs)
chairperson(carlos, hartmann, eecs, 1996, current)

What’s a current? (Don’t answer that.) The problem with this approach is that every different question we ask about someone’s property of being a chairperson adds a new argument position to the chairperson predicate to hold an answer. Each new version of chairperson would require one or more
new axioms to logically determine the relationship among the versions of the new version of the chairperson predicate in a knowledge base such as for example, a logic program or answer-set program. We would like to restrain the proliferation of complex logical constraints, and predicates in such a way as to allow for a potentially unbounded number of details. One tactic for dealing with this problem is to reify the idea of an administrative role as an abstract individual. Consider

\begin{verbatim}
admin(role137)
firstName(role137,carlos)
lastName(role137,hartmann)
type(role137,chairperson)
tenure(role137,interval96)
intervalType(interval96,rightUnbounded)
start(interval96,1996)
\end{verbatim}

**Do:** Identify as many of the reifications in the above representation as you can.

4. How do we say that the chairperson is actually unique, at least after we pin down the context we are speaking about sufficiently? The answer is the *Theory of Definite Descriptions.*

---

Bertrand Russell
1872-1970
**Nationality:** British

**Group Alliances:**
*Antagonistic* Analytic Philosophers

*Loathsome* Logicians

**AKA:** Hurtin' Bertrand
Russell the Logic Muscle
Russell "Wanna Tussle?"
The Principal of Mathematics
The Third Earl Russell

**Powers:** comprehension of sets

**Weaknesses:** radical political views about war and nuclear weapons being bad

**Notes:** Mail in a proof of purchase for this toy (plus $4.95 S & H) and receive either a Present King of France® action figure with removable toupee or a copy of *Waverly by Scott.*

Beck

21
1. The following material is adapted from Blair, Marek & Remmel, *Set Based Logic Programming*.

2. See e.g. Wolfram’s [Mathworld](http://mathworld.wolfram.com) for background on lattices.

3. **Just read this item to get a first-impression of where we are headed:**

   We introduce a lattice-based semantics for set-based programming. In the set-based stable model semantics we previously presented, we determined whether an atom $A$ is satisfiable by an interpretation $J$ (a subset of $X$) by checking to see whether $J$ contained the sense of $A$. The sense of $A$ was a subset of a fixed overlying set $X$. We then used 3-valued truth functions to extend the satisfaction relation from literals to all formulas. We did not attempt to extend the notion of *sense* to all formulas. In an algebraic semantics the sense of all formulas is defined. Conceptually, the elements of an algebraic interpretation for a logic are often thought of as abstract truth values, but this is not multi-valued logic as it is usually thought of because the number and relationship among the truth values vary from one algebraic interpretation to another. The elements of an algebraic semantics are better thought of as *sense*-values. We do **not** need to review our previous approach to understand what follows.

4. A completely elementary but very important observation concerning our set-based semantics is the following upper section property for atoms: If $A$ is an atom, then, if $J \models A$ and $K \supseteq J$, then $K \models A$. We are going to use a miop (monotone idempotent operator) to extend the notion of sense to all formulas while maintaining the upper section property. For type I and type II semantics, the upper section property holds only for atoms; **not** for all formulas. For type III semantics the upper section property holds for all formulas. First, we need to collect a few basic observations concerning lattices and miops.

5. Let $\mathcal{L}$ and $\mathcal{M}$ be posets, i.e. a set $S$ together with a partial ordering $\sqsubseteq_S$ on $S$ is a poset, and let $\varphi : \mathcal{L} \rightarrow \mathcal{M}$. $\varphi$ is **monotone** iff for all $x$ and $y$ in $\mathcal{L}$, if $x \sqsubseteq_L y$, then $\varphi(x) \sqsubseteq_M \varphi(y)$.

6. Let $\mathcal{L}$ be a poset, and let $\varphi : \mathcal{L} \rightarrow \mathcal{L}$. $\varphi$ is **idempotent** iff for all $x \in \mathcal{L}$, $\varphi(\varphi(x)) = \varphi(x)$. $\varphi$ is a **miop** (i.e monotone idempotent operator) iff $\varphi$ is both monotone and idempotent.

7. **Verify this example:** The following diagram depicts a lattice.
The partial ordering for this lattice is read this way: if there is a path from an element $x$ to an element $y$ that goes upwards, then $x \sqsubseteq y$. For example, $b \sqsubseteq \top$ and $b \sqsubseteq c$, but $b \not\sqsubseteq a$. Just by definition every element is below itself. The symbol $\sqsubseteq$ is read as ‘is below’.

8. Consider $f : \mathbb{N}_5 \rightarrow \mathbb{N}_5$ where $f(\bot) = \bot$, $f(b) = f(c) = f(a) = a$ and $f(\top) = \top$. **Verify:** $f$ is a miop.

9. Here is a very different kind of lattice: Consider the set of real numbers $\mathbb{R}$. An open interval of real numbers is a set $\{x \mid a < x < b\}$. For example the set of all real numbers greater than 0 and less than 1 is an open interval. Now consider any set of real numbers that can be formed by intersecting two open intervals. **Investigate:** Is the intersection of two open intervals always either empty, or an open interval?

10. Now consider any set that can be formed by taking the union of a collection (possibly an infinite collection) of open intervals. Sets formed form unions of open intervals are called open sets. (The empty set is considered to be the union of an empty collection of intervals.) **Prove:** The intersection of two open sets is an open set and the union of an arbitrary collection of open sets is an open set.

11. **Prove:** The set of real numbers is itself an open set.
12. **Prove:** The collection of all open sets of real numbers ordered by inclusion (i.e. the subset relation, $\subset$) is a lattice, which we denote by $\mathcal{E}$.

13. **Prove:** Let $\text{Interior} : \mathcal{E} \rightarrow \mathcal{E}$ be defined by

$$\text{Interior}(A) = \bigcup\{U \subseteq A \mid U \text{ is open}\}$$

The set $\text{Interior}(A)$ is the largest open subset of $A$.

**Prove:** Interior is a miop.

14. Let $\text{op} : 2^X \rightarrow 2^X$ be a miop.

---

**Material for: March 8, 2006**

1. **Proposition:** Let $\mathcal{Y}$ be a family of subsets of $X$ (i.e. $\mathcal{Y} \subseteq 2^X$) closed under taking $\text{op}$ of intersections and unions of subfamilies of $\mathcal{Y}$. Then the range of $\text{op}$ is a complete lattice: meet is $\text{op}$ of intersections, join is $\text{op}$ of union. Specifically, if

   (a) $\mathcal{Y} \subseteq \text{range}(\text{op})$,
   (b) for any $\mathcal{S} \subseteq \mathcal{Y}$, the join of all the elements of $\mathcal{S}$, $\bigvee_{\text{op}} \mathcal{S}$, is $\text{op}(\bigcup \mathcal{S})$, and the meet of all the elements of $\mathcal{S}$, $\bigwedge_{\text{op}} \mathcal{S}$, is $\text{op}(\bigcap \mathcal{S})$
   (c) $\bigvee_{\text{op}} \mathcal{S} \in \mathcal{Y}$, $\bigwedge_{\text{op}} \mathcal{S} \in \mathcal{Y}$

then the algebraic structure $(\mathcal{Y}, \bigvee_{\text{op}}, \bigwedge_{\text{op}})$ is a complete lattice.

**Proof:** (a) and (c) together imply that It suffices to note that (1) for finite $\mathcal{S}$ the join and meet defined above satisfy the absorption identities, and (2) for infinite $\mathcal{S}$ the join and meet of $\mathcal{S}$ are respectively, the least upper bound and greatest lower bound of $\mathcal{S}$, with respect to the ordering defined by

$$Y_1 \leq Y_2 \text{ iff } Y_1 \wedge Y_2 = Y_1.$$

2. We call the lattice given by the previous proposition the $\text{op}$-lattice over $\mathcal{Y}$, and denote it by $\mathcal{L}_{\text{op}}(\mathcal{Y})$. **Note that** $\mathcal{Y}$ must have certain closure properties for $\mathcal{L}_{\text{op}}(\mathcal{Y})$ to be defined.

3. For each subset $Y$ of $X$, let $\overline{Y} = X - Y$, and let $\overline{\mathcal{Y}} = \{\overline{Y} \mid Y \in \mathcal{Y}\}$. We abbreviate $\mathcal{L}_{\text{op}}(\text{range}(\text{op}))$ to $\mathcal{L}_{\text{op}}$. In any lattice $(X, \vee, \wedge)$, where $\leq$ is defined by $x \leq y$ iff $x \wedge y = x$, the absorption identities, $x \vee (x \wedge y) = x$ and $x \wedge (x \vee y) = x$, ensure that $x \wedge y = x$ iff $x \vee y = y$. We write $x \geq y$ for $y \leq x$ whenever convenient.
4. **Definition** The dual $o p^d$ of $o p$ is defined by

$$o p^d(I) = \overline{o p(I)}.$$ 

5. The dual of the interior operator on subsets of a topological space is called the \textit{closure} operator. A closed set is a compliment of an open set. The result of applying the interior operator $Int$ to a set $S$ is the \textit{interior} of the $S$. It is the largest open subset of $S$. The result of applying the closure operator to $S$ is the \textit{closure} of $S$.

6. We will work through an example: The 4-cube that we looked at as an example of a Lindenbaum algebra, is also the powerset lattice of a set with four elements. Take the last of the 4-cube diagrams with formulas labeling the vertices and re-label the vertices with the subsets of the set \{0, 1, 2, 3\} in such a way that the formulas $p \land p$, $p \land q$, $p \land q$ and $p \land q$ correspond to the sets $\emptyset$, \{0\}, \{1\}, \{2\} and \{3\}, respectively. Compare the set labels to the output column of the I/O table for the Boolean function defined by the formula label throughout the lattice. To see what that last sentence means, look, for example, at the vertex labeled by $p \leftrightarrow q$. The truth table for this formula label is

<table>
<thead>
<tr>
<th>p</th>
<th>q</th>
<th>$p \leftrightarrow q$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

The set that should label this vertex is \{0, 3\}. The formula $p \leftrightarrow q$ defines a Boolean function from pairs of Boolean values to Boolean-values whose I/O table is the above truth table, assuming 0 represents \textit{false} and 1 represents \textit{true}. Notice that the rows of the truth-table that have a 1 in the output column have 00 and 11 in the input columns. 00 and 11 are the binary representations of 0 and 3 respectively. The correspondence continues throughout the lattice.

7. Fully labeled, we get
1. It’s convenient to give the lattice we are working with a name: \( \text{BA}_4 \). The \textbf{BA} part stands for \textit{Boolean Algebra}. We are now going to obtain a \textit{topological sublattice} of \( \text{BA}_4 \).

2. The way to find a topological sublattice of any complete Boolean Algebra is to begin with a nonempty set of elements (i.e. vertices of the Hasse diagram). Any nonempty set will do. That set \( \mathcal{B} \) is called the \textit{subbasis}. Then downward close the subbasis with respect to pairwise meets. This means to find the smallest subset \( S \) of the Boolean Algebra that contains the subbasis such that for any two elements in \( S \), their meet is also in \( S \). The resulting set is called the \textit{basis}. Next, upward close the basis under all joins, finite and infinite. The resulting
structure, $\mathcal{T}$, is (an upward complete) distributive sublattice of the Boolean Algebra. It is a topological lattice. The elements in $\mathcal{T}$, when they are subsets in a power set lattice, are called the open sets of $\mathcal{T}$. When the elements of $\mathcal{T}$ are subsets of a given set $X$, this collection of subsets of $X$ is called a topology on $X$.

3. Returning to the specific case of $\text{BA}_4$, suppose we take as a subbasis the collection of sets, $\mathcal{S} = \{ \{1, 3\}, \{2, 3\} \}$. Closing this collection under pairwise meets we get the substructure $\text{Top}_S$ of $\text{BA}_4$ depicted below. (Verify.)

![Diagram of Top$_S$]

The substructure $\text{Top}_S$ depicted above is a lattice even standing alone (see below). It is an example of a topological lattice: a lattice $\mathcal{L}$ is topological iff it is isomorphic to a complete sublattice $\mathcal{L}'$ of a Boolean Algebra $\mathcal{B}$, where the infinite joins of $\mathcal{L}'$ agree with the infinite joins of $\mathcal{B}$ (but the infinite meets of
\( \mathcal{L}' \) do not necessarily agree with the infinite meets of \( \mathcal{B} \). What makes \( \text{Top}_S \) a sublattice of \( \text{BA}_4 \) is that the finite meets and joins in \( \text{Top}_S \) agree with the finite meets and joins of \( \text{BA}_4 \).

4. As a stand-alone structure, \( \text{Top}_S \) looks like

![Diagram](image)

This lattice is isomorphic to (which means that it has the same form as) \( \text{Top}_S \) embedded in \( \text{BA}_4 \). The isomorphism here preserves meets and joins but otherwise ignores the relative placement of the vertices as points in the plane.

5. The sets labeling the vertices of \( \text{Top}_S \) are the open sets of the topology determined this topological lattice. The interior of a set \( A \) is the largest open set contained in \( A \). In terms of the sets that label the vertices of \( \text{BA}_4 \), and the topological lattice \( \text{Top}_S \) contained within \( \text{BA}_4 \), the interior of a set \( A \) labeling a vertex \( v \) in \( \text{BA}_4 \) is the set labeling the maximum vertex in \( \text{Top}_S \) underneath \( v \). For example, the interior of \( \{0, 1, 3\} \) is \( \{1, 3\} \). **Do:** Use the figure depicting \( \text{Top}_S \) to determine the interior of each subset of \( \{0, 1, 2, 3\} \).

6. Diagrammatically, we can depict the interior of each subset of \( \{0, 1, 2, 3\} \) with:
The orange pointers in the above figure depict the interior operator relative to the chosen topology represented by $\Top_S$. For example, the orange pointer running from vertex $\{0, 1, 3\}$ to vertex $\{1, 3\}$ depicts that the interior of $\{0, 1, 3\}$ is $\{1, 3\}$. Compare: the pointer representations that indicate the interiors of each of subsets $\{0, 1, 2, 3\}$ to the interiors that you obtained in the previous item.

7. Look at the figure that depicts $\Top_S$ embedded in $\BA_4$. Use the definition of the dual of an operator to calculate the the result of applying the dual of the interior operator to each of the subsets of $\BA_4$ and show each step. For example:

\[
\{0, 3\} \xrightarrow{\text{complement}} \{1, 2\} \xrightarrow{\text{interior}} \emptyset \xrightarrow{\text{complement}} \{0, 1, 2, 3\}
\]

The above sequence of operations applied to the set $\{0, 3\}$ results in the set $\{0, 1, 2, 3\}$. Recall that the dual of an interior operator is called a closure operator. Thus, the closure of $\{0, 3\}$ is $\{0, 1, 2, 3\}$ (relative to the topology we have chosen).
8. **Label** the range of the \( \text{interior}^d \) operator. Compare \( \text{range} (\text{interior}^d) \) to \( \text{Top}_S \).

9. **Draw** pointers in \( \mathbf{B} \mathbf{A}_4 \) to depict the closure of each subset of \( \{0, 1, 2, 3\} \). Here is a quick way to do that: Take the figure for \( \mathbf{B} \mathbf{A}_4 \) above with the orange pointers that show the interior of each set, and rotate it 180°. We get:

Notice that what we get (ignore the orange pointers for a moment) is equivalent to what we would get if we just replaced each vertex label by its complement. For example, \( \{0, 2, 3\} \) has been replaced by \( \{1\} \) (although the latter label is rotated through 180°). Now go back to paying attention to the orange pointers. They all still point at the interiors of the sets they start from. For example, the orange pointer that starts from \( \{1\} \) still points to the empty set, which is the interior of \( \{1\} \). So, the vertex that was originally labeled by \( \{0, 2, 3\} \) now points to vertex labeled by the emptyset. The overall effect is that the 180° rotation is mapping each set to its complement, and the orange pointer now maps that complement to *its* interior. If we now replace all the vertex labels...
by their original labels we are effectively applying the complement operator one
last time to the result accumulated so far. That’s the same as the dual of the
interior operator! We get

Material for: March 22, 2006

1. The $\mathbb{N}_5$ structure inside $\mathbb{B}A_4$ looks like:
The miop on $\mathbf{BA}_4$ that we studied last time, was

$\rho$

$\mathbf{BA}_4$ with $\mathbf{N}_5$
**Calculate** $\rho^d$ and display its range within $\mathbf{B}A_4$. Verify that you get for the range of $\rho^d$.

and verify that for $\rho^d$ you get
<table>
<thead>
<tr>
<th>input</th>
<th>output</th>
</tr>
</thead>
<tbody>
<tr>
<td>∅</td>
<td>∅</td>
</tr>
<tr>
<td>{0}</td>
<td>∅</td>
</tr>
<tr>
<td>{1}</td>
<td>{1}</td>
</tr>
<tr>
<td>{2}</td>
<td>∅</td>
</tr>
<tr>
<td>{3}</td>
<td>∅</td>
</tr>
<tr>
<td>{0, 1}</td>
<td>{1, 2}</td>
</tr>
<tr>
<td>{0, 2}</td>
<td>∅</td>
</tr>
<tr>
<td>{0, 3}</td>
<td>∅</td>
</tr>
<tr>
<td>{1, 2}</td>
<td>{1, 2}</td>
</tr>
<tr>
<td>{1, 3}</td>
<td>{1, 2}</td>
</tr>
<tr>
<td>{2, 3}</td>
<td>{2, 3}</td>
</tr>
<tr>
<td>{0, 1, 2}</td>
<td>{1, 2}</td>
</tr>
<tr>
<td>{0, 1, 3}</td>
<td>{1, 2}</td>
</tr>
<tr>
<td>{0, 2, 3}</td>
<td>{2, 3}</td>
</tr>
<tr>
<td>{1, 2, 3}</td>
<td>{1, 2, 3}</td>
</tr>
<tr>
<td>{0, 1, 2, 3}</td>
<td>{0, 1, 2, 3}</td>
</tr>
</tbody>
</table>

2. Proposition:

(a) \( op^{dd} = op \)

(b) \( \text{range}(op^d) = \text{range}(op) \).

Verify both parts of this proposition where \( op \) is the interior operator, using the example of the topological sublattice \( \text{Top}_S \) of \( \text{BA}_4 \).

3. Proposition: Let \( \mathcal{Y} \) be a family of subsets of \( X \) (i.e. \( \mathcal{Y} \subseteq 2^X \)) closed under taking \( op \) of intersections and unions of subfamilies of \( \mathcal{Y} \). Then the following DeMorgan Principles hold:

(a) For each \( S \subseteq \mathcal{Y} \), \( \bigvee_{op}(\mathcal{S}) = \bigwedge_{op}(\mathcal{S}) \), and

(b) \( \bigwedge_{op}(\mathcal{S}) = \bigvee_{op}(\mathcal{S}) \).

Verify that the proposition concerning the Demorgan principles is true when applied to the example of the topological sublattice \( \text{Top}_S \) of \( \text{BA}_4 \). Again, take \( op \) to be the interior operator.

Material for: March 22, 2006
1. Note also that an \( op \)-lattice need not be distributive. Verify that the \( N_5 \) lattice within \( BA_4 \), is the \( \rho \)-lattice, corresponding to the miop \( \rho \) that we gave in the input/output table above. FYI: Topology plays a significant role in logic. One factor contributing to the role topology plays in logic is that the interior miop \( Int \) has the correct distributivity properties itself to yield a distributive \( Int \)-lattice.

2. Verify that \( N_5 \) is not distributive.

3. Verify, again using \( N_5 \) lattice within \( BA_4 \) and taking \( \rho \) for \( op \), that if \( \mathcal{V} = \text{range}(op) \), then \( op(\emptyset) \) is the bottom of the \( op \)-lattice, and \( op(X) \) is the top.

4. The simple dual of a lattice is obtained by interchanging meet and join. Observe that the simple dual of an \( op \)-lattice has meet as the \( op \) of unions and join as the \( op \) of intersections.

5. Summing up the apparatus we have collected so far, we get the following definition:

**Definition** The De Morgan dual of an \( op \)-lattice \( \mathcal{L}_{op}(\mathcal{V}) \) is the \( op^d \)-lattice \( \mathcal{L}_{op^d}(\mathcal{Y}) \).

The coherence of this definition raises questions that we have already answered. In particular: \( op^d \)-lattice \( \mathcal{L}_{op^d}(\mathcal{Y}) \).

6. **Proposition:** The Demorgan dual of \( op \)-lattice \( \mathcal{L}_{op}(\mathcal{V}) \) is isomorphic to the simple dual of \( op \)-lattice \( \mathcal{L}_{op}(\mathcal{V}) \).

**Material for:** March 29, 2006

1. We denote the De Morgan dual of \( \mathcal{L}_{op} \) by \( \mathcal{L}_{op^d} \). **Reminder:** the sets that are the elements of \( \mathcal{L}_{op^d} \) are the complements of the sets that are the elements of \( \mathcal{L}_{op} \).

2. The lattice-based approach to the semantics of a spatially augmented language \( \mathcal{L} \) begins with senses of atoms chosen to be closed with respect to a given miop, and then extends the sense assignment inductively. We assume that we have a spatially augmented language in which the sense \( [A] \) of an atom \( A \) is closed with respect to a miop \( op \). We now extend the sense notion to all formulas. The sense of a formula is an element of the lattice \( \mathcal{L}_{op} \), which in general lacks complements. In the general absence of complements the semantics of the conditional is not simply reducible to a composition of negation and disjunction. **Do:** Show that in ordinary propositional logic the conditional is reducible to a composition of negation and disjunction.
Definition: (Senses of closed formulas) We denote the sense of a closed formula $\varphi$ by $[\varphi]$. (A formula is closed iff it does not contain any free occurrence of a variable; i.e. every variable in the formula occurs within the scope of a quantifier for that variable.) We are assuming that the senses of the atomic formulas have been given.

(a) $[A \lor B] = [A] \land [B]$
(b) $[A \land B] = [A] \lor [B]$
(c) $[A \rightarrow B] = \bigwedge\{W \in \mathcal{L}_{op} \mid (W \lor [A]) \geq [B]\}$
(d) $[\neg A] = \bigwedge\{W \in \mathcal{L}_{op} \mid (W \lor [A]) = \text{op}(X)\}$
(e) $[\forall x A(x)] = \bigvee\{[A(t)] \mid t \text{ a variable-free term}\}$
(f) $[\exists x A(x)] = \bigwedge\{[A(t)] \mid t \text{ a variable-free term}\}$

3. Example: Two of the six vertices in the lattice below are labeled by formulas. Using the definition of the sense of a formula above, find: formula labels that use no Boolean variables other than $p, q$ for the remaining six vertices. In particular, what vertices do $p \rightarrow q$ and $\neg p$ label? ($\text{op}(X)$ is the top element in the lattice.)

4. We can define the dual-$\rightarrow$ operation on a lattice as follows:

$$x \rightarrow_d y = \bigwedge\{w \mid w \lor x \geq y\}.$$
5. For a lattice with a bottom element, define the dual pseudo-complement operation \( \neg^d \) on the lattice by 
\[
\neg^d x = x \rightarrow^d \bot.
\]

6. **Example:** Use the \( N_5 \) lattice below to calculate the dual pseudo-complement of every element, and the dual-\( \rightarrow \) operation on every pair of elements.

7. **Definition:** (Lattice-based semantics) Let \( L \) be a lattice, and assume that the sense of each atom has been chosen as an element of \( L \). For any element \( I \) of \( L \), \( I \models L A \iff [A] \subseteq I \).

8. The next goal is to give the dual lattice-based semantics. A Heyting algebra (cf. Troelstra and van Dalen, 1988 (TvD88)) is a lattice with a bottom element, together with an additional operation \( \rightarrow \) that satisfies
\[
(x \land y) \leq z \iff x \leq (y \rightarrow z)
\]
As a consequence of the definition of Heyting algebra, **claim 1:** a Heyting algebra is a distributive lattice in which meet distributes over both finite and infinite joins whenever the infinite joins are defined. **Claim 2:** Any complete lattice in which meet distributes over all joins (finite and infinite) is extendable to a (complete, because the lattice substructure is complete) Heyting algebra where
\[
x \rightarrow y = \bigvee \{z \mid z \land x \leq y\}.
\]
(Graduate students should verify the two claims.)
9. **Definition:** A *topological* complete Heyting algebra is a Heyting algebra in which the elements are open sets in a topology, join is union, meet is the interior of (needed for infinitary intersections) intersection, and \( \rightarrow \) is defined as indicated immediately above cf. TvD88.

10. **Definition:** *(Dual senses of closed formulas)* For each formula \( A \),
    \[
    [A]^d = \overline{[A]}. 
    \]

11. **Definition:** *(Dual lattice-based semantics)* Let \( L \) be a lattice. For any element \( I \) of \( L \),
    \[
    I \models_L A \text{ iff } I \subseteq [A]^d. 
    \]

12. **Proposition:**
    (a) If \( A \) is an atom, \([A]^d = \overline{[A]} \).
    (b) \([A \lor B]^d = \overline{[A \land [B]]} = [A]^d \lor_{op^d} [B]^d \).
    (c) \([A \land B]^d = \overline{[A \lor [B]]} = [A]^d \land_{op^d} [B]^d \).
    (d) \([A \rightarrow B]^d = \mathop{\bigvee}_{op^d} \{W \in \mathcal{L}_{op^d} \mid (W \land_{op^d} [A]^d) \leq_{op^d} [B]^d\}\}
    (e) \([-A]^d = \mathop{\bigvee}_{op^d} \{W \in \mathcal{L}_{op^d} \mid (W \land_{op^d} [A]^d) = op(\emptyset)\}\}
    (f) \([\forall x A(x)]^d = \mathop{\bigwedge}_{op^d} \{[A(t)]^d \mid t \text{ a variable-free term}\}\}
    (g) \([\exists x A(x)]^d = \mathop{\bigvee}_{op^d} \{[A(t)]^d \mid t \text{ a variable-free term}\}\}
    (h) Whenever \( \mathcal{L} \) is an \( op \)-lattice, the bottom of \( \mathcal{L}_{op^d} \) is \( op^d(\emptyset) \). The top is \( op^d(X) \).

13. **Do:** Calculate the Demorgan dual of \( \text{Top}_S \) within \( \text{BA}_4 \). Temporarily, we will call this Demorgan dual \( L_0 \). \( L_0 \) has the same structure as
but is embedded within $\mathbf{BA}_4$. Within $\mathbf{BA}_4$ label the vertices of $L_0$ with $p$ and $q$ as shown just above. Identify the vertices of $L_0$ that correspond to $[p \land q]$, $[p \lor q]$, $[p \rightarrow q]$, and $[\neg p]$.

14. In $\mathbf{Top}_S$ within $\mathbf{BA}_4$, calculate the dual senses of the formulas $[p \land q]$, $[p \lor q]$, $[p \rightarrow q]$, and $[\neg p]$.

15. Verify, separately, that both $\mathbf{BA}_4$ and $\mathbf{Top}_S$ are Heyting algebras, but that $\mathbf{N}_5$ is not, by showing how the Heyting algebra axiom fails for $\mathbf{N}_5$.

Material for: April 5, 2006
Spatial Logic Programming

Victor W. Marek

University of Kentucky

Presentation at the University of Texas, Spring Semester 2006

Joint work with H.A. Blair and J.B. Remmel
Two fundamental approaches to Logic Programming:

- Logic-based (with some logical formalism and its processing strategies)
- Operator-based (computation of fixpoints of some operators)
Logic-based approaches

- Prolog (Horn Logic and its extensions, Kowalski, Colmerauer)
- Answer Set Programming (Stable semantics of programs, Niemelä, Lifschitz, M-T)
  - Spectacular achievements here (Pearce, Lifschitz and collaborators)
  - Almost complete understanding of ASP as we know it
  - Practical implementations (smodels, cmodels, dlv, assat, nomore)
- Monotone operators in complete lattices provide semantics for Horn logic
- Antimonotone operators provide semantics for ASP
- Approximation operators (Denecker, M. and Truszczynski) explain multivalued stable models
- Can we do more with operators?
- Building blocks (like one on the left)
- Building figures from building blocks
Example, cont’d

I would like to write programs like this:

\[
\begin{align*}
square(0, 0, 1) \\
square(1, 1, 1) \\
square((X_1 + X_2)/2, (Y_1 + Y_2)/2, (Z_1 + Z_2)/4) &← \\
square(X_1, Y_1, Z_1), \ square(X_2, Y_2, Z_2) \\
result(X, Y) &← \ square(X_1, Y_1, 1) \\
result(X, Y) &← \ square(X_1, Y_1, 1/2)
\end{align*}
\]

With results as in the picture we have seen above

Possibly also with negation in the body
Operators in lattices

- An operator in a lattice \( \langle L, \leq \rangle \) is any mapping \( O \) from \( L \) to \( L \)
- \( O \) is idempotent if \( O(O(x)) = O(x) \)
- \( O \) is monotonic if \( x \leq y \Rightarrow O(x) \leq O(y) \)
- Examples: closure, interior, span, convex-closure are all monotone idempotent
- \textit{miop} – a monotone idempotent operator
- \( X \) is closed under miop \( O \) if \( X = O(X) \) (i.e. is a fixpoint)
- Various intuitions associated with miops
Sense of an atom

- $I \models p$ if $p \in I$
- A sense of atom $p$, $[p]$ a subset of some space
- In normal logic programming $[p] = \{p\}$
- Then $I \models p$ if $[p] \subseteq I$
- Now we have $[p] \subseteq S$ a sense of $p$ in space $S$
- An interpretation is now a subset of $S$
- Satisfaction inherited from LP:
  \[ I \models p \text{ if } [p] \subseteq I \]
New possibilities for negation

- In Normal LP, a model $M$ satisfies $\neg p$ if $p \notin M$
- But here we have a variety of possibilities generalizing two different formalizations
  - One possibility is: $M \models \neg p$ if $\{p\} \cap M = \emptyset$
  - Another possibility: $\{p\} \not\subseteq M$
  - Miops open additional possibilities
- Those generalize in the current setting in two ways:
  - $I \models \neg p$ if $\llbracket p \rrbracket \cap I = \emptyset$
  - $I \models \neg p$ if $\llbracket p \rrbracket \not\subseteq I$
  - Thus there are two kinds of NAF, and two potential notions of a stable model
What happened till now?

- As long as we have Horn programs (no negation) without miops all we do is least fixpoint computation and then remember that atoms have senses and we just take union of those senses
- But we already see that once non trivial senses are allowed we immediately get something more
- miops will bring more possibilities
Negation possibilities with miops

- Given a miop $\text{op}$
  - One possibility is $I \models \neg p$ if $\text{op}([p]) = \emptyset$
  - Another possibility is $I \models \neg p$ if $\text{op}([p]) \not\subseteq I$
  - But with miops $\text{op}(\emptyset)$ may be nonempty, so we will have to be more careful
The language is $\langle L, X, [\cdot]\rangle$

- $L$ is FOL, no function symbols
- $X$ is the interpretation space
- $[\cdot]$ maps ground atoms of $L$ into powerset of $X$, $[p]$ is the sense of $p$
- We need to be careful when miops are used, and satisfaction needs to make sure that the interpretations are $op$-closed
Spatial Logic Programs

- Spatially augmented language
- Intentional database, a finite set of program clauses each of the form $A \leftarrow L_1, \ldots, L_n$, where each $L_i$ is a literal, i.e. an atom or the negation of an atom, and $A$ is an atom
- Extensional Database a finite set of ground atoms
- (For a moment no miops but they will come in later)
Satisfaction relations

- Since we can have a variety of negations, we can introduce a variety of satisfaction relations.
- Interpretations: subsets of $X$.
- The sense function $[\cdot]$ determines satisfaction for ground atoms (but recall caveat of what are interpretations - they need to be called under $op$).
- But we may have various negations, and so we can have various satisfaction relations (each definition of satisfaction for negation induces its own satisfaction relation for the entire language).
  - (In fact a similar technique was used by Lifschitz in MBNF)
Models and operators

- A model of a program - interpreting clauses as universally quantified conditionals
- Operator $T_P$
- $T_P(I) = I_1 \cup I_2$

$$I_1 = \bigcup \{ [A] \mid A \leftarrow L_1, \ldots, L_n \in P', I \models L_i, i = 1, \ldots, n \}$$

$$I_2 = \bigcup \{ [A] \mid A \text{ is a ground atom in the EDB of } P \}.$$  

- As usual, a model is a prefixpoint of $T_P$
- A supported model of $P$ is defined as a fixpoint of $T_P$
- There were no miops, we can think that the miop is identity operator
Horn programs, least fixpoint

- **Horn programs: IDB Horn**
- A least model of a Horn program exists, is supported and given by some iteration of $T_P$
- If Herbrand universe is finite fixpoint reached in finite number of steps
- Otherwise infinite number of steps may be required
As in the case of NLP the notion of negation induces Gelfond-Lifschitz transform w.r.t interpretation $J$

Given $r := A \leftarrow B_1, \ldots, B_{12}, \neg C_1, \ldots, \neg C_{13}$

- Eliminate clause $r$ with $J \models C_i$ for some $i$
- Drop $C_i$s otherwise

Get the reduct $GL(J, P)$

$J$ is stable model of $P$ if $J$ is the least model of the reduct
Some properties of models so obtained

► Stable models, as in NLP case are supported
► If $\mathcal{L}$ is a spatially augmented first-order language, then the set of senses of the ground atoms form a subbasis of a topology in which all supported models, a fortiori all stable models, of all spatial programs over $\mathcal{L}$ are open subsets of the interpretation space.
Examples of miops

1. Our goal is to incorporate miops (so, for instance, we can eliminate boundary in our picture)
2. $\text{op}_{id}(A) = A$, i.e. the identity map is simplest miop operator,
3. $\text{op}_{c}(A) = \overline{A}$ where $\overline{A}$ is the smallest closed set containing $A$,
4. $\text{op}_{int}(A) = \text{int}(A)$ where $\text{int}(A)$ is the interior of $A$,
5. $\text{op}_{\text{convex}}(A) = K(A)$ where $K(A)$ is the convex closure of $A$, i.e. the smallest set $K \subseteq X$ such that $A \subseteq K$ and whenever $x_1, \ldots, x_n \in K$ and $\alpha_1, \ldots, \alpha_n$ are elements of the underlying field $R$ or $Q$ such that $\sum_{i=1}^{n} \alpha_i = 1$, then $\sum_{i=1}^{n} \alpha_i x_i$ is in $K$ (requires some linear structure)
6. $\text{op}_{\text{subsp}}(A) = (A)^{\ast}$ where $(A)^{\ast}$ is the subspace of $X$ generated by $A$ (linear spaces needed)
Given a miop $\text{op}^+ : 2^X \rightarrow 2^X$ and spatial logic program $P$

$$T_{P,\text{op}^+}(I) = \text{op}^+(I_1 \cup I_2)$$

- $I_1$ is the union of senses of heads whose bodies are satisfied
- $I_2$ is the union of EDB
- We can, of course iterate this operator
- We can introduce a concept of a supported models w.r.t a miop
- Horn programs, as before – IDB Horn
Least model with miop

- Given a miop $op^+$, the least model $I$ of spatial Horn program $P$ which is closed under $op^\pm$ exists
- $I$ is supported relative $op^+$
- $I$ is given by $T_{P,op^+} \uparrow^\alpha (\emptyset)$ for the least ordinal $\alpha$ at which a fixpoint is obtained.
How to interpret negation, stability

- Given any binary relation $\mathcal{R}$ on subsets of the space $X$
- $J \models^\mathcal{R} \neg A$ if $\mathcal{R}(J, \llbracket A \rrbracket)$
- Now let $r := A \leftarrow A_1, \ldots, A_m, \neg B_1, \ldots, B_n$
  - If for some $i$, $1 \leq i \leq n$ $J \models^\mathcal{R} \neg C_i$ eliminate $r$
  - O/w drop negated atoms from $r$, i.e. simplify $r$
- Thus the program $GL_{J,\mathcal{R}}(P)$ obtained
- Now, let us assume that we have a miop $op^+$
- $J$ is a stable model of $P$ w.r.t $op^+$ and $\mathcal{R}$, if $J$ is the least fixpoint of the operator $T_{GL_{J,\mathcal{R}(P)},op^+}$
We may have a different miop $op^-$ on the negative side

If $P$ is Horn, closure only under $op^+$

But if we have this additional miop $op^-$ then we can define two negations:

- $\mathcal{R}_1(J, K)$ if $J \cap op^-(K) = op^-(\emptyset)$
- $\mathcal{R}_2(J, K)$ if $op^-(K) \not\subseteq J$
Then we have two definitions of satisfaction (the difference being in negation):
- $J \models^{I}_{op-} \neg A$ if $op^{-}(\mathcal{A}) = \mathcal{A}^{-}(\emptyset)$
- $J \models^{II}_{op-} \neg A$ if $op^{-}(\mathcal{A}) \not\subseteq J$

We get two associated operators in space $X$
- $T^{I}_{P,op^{+},op^{-}}$
- $T^{II}_{P,op^{+},op^{-}}$
Bringing second miop, two stable semantics

- We get two Gelfond-Lifschitz operators
- and we get two types of stable models that we call type I and type II stable models
Sample of results

- Observations:
  - Let us observe that if $op^-=op_{id}$ and the sense of each negated atom occurring in the program is a singleton, then type I and type II stable models coincide.
  - By varying miops a single program may be used to encode different concepts.

- Separating sets for a pair of sets $A, B$. By choice of different miops with the same program we can express several types of separating sets as stable models (in this programs senses are singletons, so it does not matter which type we choose):
  - Separating sets, i.e. $C$ such that $A \subseteq C$ and $B \subseteq V \setminus C$.
  - Closed separating sets, i.e. closed $C$ such that $A \subseteq C$ and $B \subseteq V \setminus C$.
  - Open separating sets, i.e. open $C$ such that $A \subseteq C$ and $B \subseteq V \setminus C$.
  - Convex separating sets, i.e. convex $C$ (with convex complement) such that $A \subseteq C$ and $B \subseteq V \setminus C$. 
More on separation

- With a different program, and with a suitable choice of miops we can represent as stable models (again of type I which is same as type II in this case) complementary subspaces
Distinguishing type I and type II stable models

- There are finite propositional programs where senses of atoms are two-element, where type I and type II stable models are different.
- That is, we can distinguish between stable models of type I and stable models of type II, and the examples where there is a difference are as simple as possible.
The concept of continuous function in reals can be captured by stable models of a suitably chosen spatial program
(a propositional program presented in the paper)
(actually, with some coding this can be done in NLP, but here it is done directly)
Conclusions

- There is a form of Answer Set Programming over spaces more complex than $\text{Bool}$.
- In this form of ASP we see that there are various GL-operators (they just collapse over $\text{Bool}$).
- Like in ASP various mathematical constructions can be expressed in this formalism, except that it appears to be connected to continuous, not only discrete mathematics.
- Hence we get a glimpse into a potentially new area of applications of ASP.
Material for: April 12, 2006


2. Try out the following Prolog example:

```prolog
move(1,From,To,Spare,towers(Left,Middle,Right),
     towers(NewLeft,NewMiddle,NewRight)) :-
    X write('Move top disk from '),
    X write(From),
    X write(' to '),
    X write(To),
    X nl.
/* Replace the X'd lines */
move(N,From,To,Spare,towers(Left,Middle,Right),
     towers(NewLeft,NewMiddle,NewRight)) :-
    N>1,
    M is N-1,
    move(M,From,Spare,To,towers(Left,Middle,Right),
         towers(Left1,Middle1,Right1)),
    move(1,From,To,_,towers(Left1,Middle1,Right1),
         towers(Left2,Middle2,Right2)),
    move(M,Spare,To,From,towers(Left2,Middle2,Right2),
         towers(NewLeft,NewMiddle,NewRight)).
```

Material for: April 17, 2006

1. Visit and use as a reference
   http://www.coli.uni-saarland.de/~kris/learn-prolog-now
   In particular, study and trace the behavior of the 3-ary reverse that is discussed in the online Prolog tutorial.

2. During the class of Wednesday, April 12th the following was assigned: Prove: If $F$ is a Kripke frame with a transitive accessibility relation $R$, then for every propositional modal formula $A$, 

$$w \models \square A \rightarrow \square \square A$$

at each world $w \in F$. 
3. Here is an example similar to the above exercise: If $F$ is a Kripke frame with a serial accessibility relation $R$, then for every propositional modal formula $A$,

$$w \models \Box A \rightarrow \Diamond A$$

at each world $w \in F$. (A relation $R$ on a set $S$ is serial if $$(\forall x \in S)(\exists y \in S)xRy.$$)

**Proof:** Let $w$ be an arbitrary world in $F$, and let $A$ be an arbitrary formula. We must prove

$$w \models \Box A \rightarrow \Diamond A.$$

It therefore suffices to prove

$$if \ w \models \Box A \ then \ w \models \Diamond A.$$

Suppose $w \models \Box A$. It remains to prove

$$w \models \Diamond A$$

which is equivalent to

$$w \models \neg \Box \neg A$$

which is equivalent to

$$w \not\models \Box \neg A$$

which holds iff there is some world $v$ such that $wRv$ where

$$v \not\models \neg A$$

holds. But

$$v \not\models \neg A \ iff \ v \models \neg \neg A \ iff \ v \models A.$$ 

Thus, it remains to prove that for some world $v \in F$ such that $wRv$,

$$v \models A.$$

$R$ is serial. Therefore, there is some world $v \in F$ such that $wRv$. We have assumed

$$w \models \Box A.$$

Therefore

$$v \models A$$

which was to be proved.
1. Refer to the Wikipedia articles on Kripke semantics and intuitionistic logic at http://en.wikipedia.org/Kripke_semantics

2. Consider the syntactic structure (i.e. the grammatical form) of the formula □A → □□A

We can always translate a propositional connective in a formula, such as V in P V Q, in terms of ¬ and →.

For example, P ∨ Q translates as ¬P → Q because the formulas P ∨ Q and ¬P → Q are equivalent, as you can see by using a truth-table.

So, we only need the two connectives ¬ and →. Every other connective can be re-expressed in terms of these. When we use the other connectives, they can be regarded as convenient “syntactic sugar” to make formulas easier to read. Modal propositional formulas also are allowed to the box symbol, which grammatically is like ¬.

This means there are four kinds of modal propositional formulas:

(a) **atoms**: i.e. Boolean variables
(b) **negations**: formulas like ¬A, where A itself is any of the four kinds of modal propositional formulas
(c) **conditionals**: formulas like A → B, where A and B are any of the four kinds of modal propositional formulas
(d) **necessities**: formulas like □A, where A is any of the four kinds of modal propositional formulas

Every modal propositional formula belongs with exactly one of the four kinds listed above.

So, **first question**: which kind of formula is □A → □□A?
1. **Topic: Autonepistemic Logic:** (cf. Section 5, chapter 11, Brachman and Levesque). **Begin by recalling** stable models of logic programs.

2. Given a logic program $P$ with clauses of the form

   \[ A \leftarrow B_1 \land B_2 \land \ldots \land B_m \land \neg C_1 \land \neg C_2 \land \ldots \land \neg C_n \]

   **stable models** of $P$ were found by guessing an Herbrand interpretation $I$ and using $I$ to assign truth values to the negative literals (like the $C_j$ that occur in the clauses of $P$) to obtain a program $P'$ without such negative literals (called the Gelfond-Lifschitz transform of $P$) and then checking whether the least Herbrand model $M$ of $P$ is $I$. If $M = I$ then $I$ is stable, otherwise not.

3. Stable models effectively interpret clauses like

   \[ A \leftarrow B_1 \land B_2 \land \ldots \land B_m \land \neg C_1 \land \neg C_2 \land \ldots \land \neg C_n \]

   as follows: if $B_1 \land B_2 \land \ldots \land B_m$ is true, and it is consistent to assume that $\neg C_1 \land \neg C_2 \land \ldots \land \neg C_n$ then conclude $A$.

4. This way of interpreting program clauses treats program clauses as what are called **defaults**. Defaults in general have the form

   \[
   \frac{\alpha : \beta}{\gamma}
   \]

   which is read,“if $\alpha$ and it is consistent to assume $\beta$, then infer $\gamma$”. A more relaxed reading is: “if $\alpha$ is believed and it is consistent to believe $\beta$ then believe $\gamma$”. This latter interpretation takes us into **autoepistemic logic**, a logic that can allow us to reason **about** defaults as well as with defaults.

5. Consider

   Any bird not believed to be flightless flies.

   We could represent this sentence in a modal logic as

   \[
   \forall x \left[ \text{bird}(x) \land \neg B \neg \text{flies}(x) \rightarrow \text{flies}(x) \right]
   \]

   The operator $B$ is grammatically like $\Box$ and $\Diamond$. The problem before us to give this operator an exact meaning so that mathematically and computationally we know what we’re talking about.

6. There are two basic approaches:
(a) stable expansions
(b) Kripke-like semantics

7. **Stable expansions:** (Analogous to stable models) Let $\mathbf{KB}$ be a set of autoepistemic sentences of first-order logic (i.e. formulas that are allowed to use the belief operator $\mathbf{B}$ and that contain no free variables).

A set $\mathcal{E}$ of autoepistemic formulas is a *stable expansion* of $\mathbf{KB}$ if, and only if the following property holds for each autoepistemic sentence $\varphi$

$$\varphi \in \mathcal{E} \iff (\mathbf{KB} \cup \{\mathbf{B}\alpha \mid \alpha \in \mathcal{E}\} \cup \{\neg\mathbf{B}\alpha \mid \alpha \notin \mathcal{E}\}) \models \varphi.$$

This condition says that $\varphi \in \mathcal{E}$ iff all sentences in $\mathbf{KB}$ together with all of the assertions that say that each sentence in $\mathbf{KB}$ is believed and all of the assertions that say that each sentence not in $\mathbf{KB}$ is not believed logically imply $\varphi$.

8. Another way to say what an *autoepistemic expansion* is, is that an autoepistemic expansion of $\mathbf{KB}$ is a minimal set $\mathcal{E}$ of autoepistemic sentences that contains $\mathbf{KB}$ as a subset and is closed under the following three principles

(a) **(logical consequence)** If $\mathcal{E}$ logically implies $\varphi$ then $\varphi$ is in $\mathcal{E}$.

(b) **(positive introspection)** If $\varphi$ is in $\mathcal{E}$ then $\mathbf{B}\varphi$ is in $\mathcal{E}$. (i.e. Every sentence in the robot’s belief base is actually is seen by the robot to be believed by the robot, or yet another way to put it: the robot knows what it thinks it knows.)

(c) **(negative introspection)** If $\varphi$ is not in $\mathcal{E}$ then $\neg\mathbf{B}\varphi$ is in $\mathcal{E}$. (i.e. If the robot doesn’t know something, then, like Socrates, it knows that it doesn’t know it.)

9. **Kripke-like semantics:** See

Logic, Advanced Course (TDDB08)

for the following:
"It is reasonable to hope that the relationship between computing and logic will be as fruitful in the next century as that between analysis and physics in the last."
(J. McCarthy, 1960)
Autoepistemic reasoning

Autoepistemic logic (Moore, 1985) is built over a base logic (e.g., the classical propositional or first-order logic) by adding modality □ to the language. The semantics of modality □ is usually given by modal logics S5 or K45. Typical forms of autoepistemic reasoning are based on rules "if a fact A is not known then conclude that ¬A holds", i.e.,

\[ \neg \Box A \rightarrow \neg A \]

— this is not an axiom of the logic (that would lead to the trivial modal logic), but is accepted for certain formulas A.

Examples

1. Recall the autoepistemic rule "if x is not on a list of winners, assume that x is a loser.". It can be expressed as:

\[ \neg \Box \text{winner}(x) \rightarrow \neg \text{winner}(x), \]

where \( \Box \text{winner}(x) = \text{TRUE} \) when x is on the list of winners.

2. Consider "if x does not know that (s)he has a sister, conclude that x has no sister" (otherwise x would know about it):

\[ \neg \Box \text{hasSister}(x) \rightarrow \neg \text{hasSister}(x). \]
11. **Look up** the modal logics K45 and S5 on the Wikipedia Kripke semantics website. Compare K45 and S5. Are they the same? Prove your conclusion. Here is a suggestion for how to go about this: K45 requires each of axiom schema K, 4 and 5 to be valid at every world in a given Kripke frame. A Kripke frame $F$ whose accessibility relation does not have the right properties to make each of K, 4 and 5 valid at every world in $F$ is rejected because K45 cannot have a Kripke model based on that frame. Every instance of axiom schema K comes out valid at every world in every Kripke frame. Axiom schema 4 requires the accessibility relation of a Kripke frame to be transitive for every instance of axiom 4 to be valid at every world in the Kripke frame. Similarly, axiom schema 5 requires the accessibility relation of a Kripke frame to be Euclidean. So, the axioms of modal logic K45 jointly require the accessibility relation of a frame that can support a model of K45 model to be both transitive and Euclidean. Similarly, modal logic S5, which is the same as modal logic T5, requires a Kripke frame in which every instance of axiom schemata T and 5 is valid at every world in the frame. In order for every instance of axiom schemata T and 5 to be valid at every world in a Kripke frame, the accessibility relation of the frame has to be reflexive and Euclidean. All of this is contained in the two tables, *Common modal axiom schemata* and *Common normal modal logics* from the Wikipedia Kripke Semantics website.

So, in order to compare modal logics K45 and T5, you just have to compare relations that are transitive and Euclidean with relations that are reflexive and Euclidean. In particular, is every relation on a set $W$ that is reflexive and Euclidean also transitive and Euclidean? Conversely, is every relation on a set $W$ that is transitive and Euclidean also reflexive and Euclidean? In other words, this exercise here in item 11 is almost the same as the exercise in the next item, item 12. In fact, the exercise in item 12 gives you the answer to the exercise in item 11.

12. **Exercise:** Prove that a reflexive Euclidean relation is reflexive, symmetric and transitive. **Investigate:** Is a transitive, Euclidean relation also reflexive?

**Supplementary Material**

The following material is from:


For Van Dalen and Troelstra the term *inhabited* means *nonempty*. Intuitionism stresses however that a set should be regarded as nonempty only if at least one of its elements can be found.
Also, the acronym PEM means “Principle of Excluded Middle”.

An expository paper on this material should fill in the details of the proof of the implicit proposition in II.11.5.6 as well as, and more importantly, the theorem in II.11.5.7. Leading up to that, the details of the proof of Proposition 4.13 should be carried out. That requires looking up what Zorn’s Lemma is all about. The rest of the material has largely been covered in the course, but is included for reference. The first two images below are from Chapter 5, Volume 1. The subsequent sections are from Chapter 11, Volume 2.

5.2. Definition. A (propositional) Kripke model is a triple \( \mathcal{X} = (K, \preceq, \Vdash) \), where \((K, \preceq)\) is an inhabited, partially ordered set (poset), and \( \Vdash \) a binary relation on \( K \times \mathcal{P} \) (the set of proposition letters) such that

\[
k \Vdash P \text{ and } k' \preceq k \Rightarrow k' \Vdash P.
\]

We shall use the expressions “\( k \) forces \( P \)” or “\( P \) is true at \( k \)” for \( k \Vdash P \).

\( \Vdash \) is then extended to logically compound formulas by the following clauses

- **Kr1** \( k \Vdash A \land B \iff k \Vdash A \) and \( k \Vdash B \),
- **Kr2** \( k \Vdash A \lor B \iff k \Vdash A \) or \( k \Vdash B \),
- **Kr3** \( k \Vdash A \rightarrow B \iff \) for all \( k' \preceq k \), if \( k' \Vdash A \) then \( k' \Vdash B \),
- **Kr4** not \( k \Vdash \bot \) (i.e. \( k \not\Vdash \bot \): no element of \( K \) forces \( \bot \)).

The elements of \( K \) are called nodes (of \( \mathcal{X} \)).

Remark. As a consequence of this definition

\[ k \Vdash \neg A \iff \forall k' \preceq k (k' \not\Vdash A). \]
Also

\[ k \vdash \neg A \iff \forall k' \geq k \forall k'' \geq k'(k'' \not\vdash A), \]

which is classically equivalent to

\[ \forall k' \geq k \exists k'' \geq k'(k'' \not\vdash A). \]

\[ \square \]

5.3. LEMMA. For all formulas of IPC we have monotonicity:

\[ \forall k, k' \in \mathcal{X} (k \vdash A \land k' \geq k \Rightarrow k' \vdash A). \]

PROOF. By formula induction. Consider for example the case \( A \equiv B \rightarrow C \).

Assume \( k \vdash B \rightarrow C \), \( k' \geq k \). If \( k'' \geq k' \), \( k'' \vdash B \), then also \( k'' \geq k \) and \( k'' \vdash B \), hence by \( k \vdash B \rightarrow C \) also \( k'' \vdash C \); thus \( \forall k'' \geq k'(k'' \vdash B \Rightarrow k'' \vdash C) \), i.e. \( k' \vdash B \rightarrow C \). \[ \square \]

5.4. DEFINITION. A formula \( A \) is valid at \( k \) in a Kripke model \( \mathcal{X} \) iff \( k \vdash A \). A is valid in \( \mathcal{X} (K, \leq, \vdash) \) iff for all \( k \in K \), \( k \vdash A \); notation \( \mathcal{X} \vdash A \). If \( \Gamma \) is a set of formulas, we say that \( \Gamma \vdash A \) ("\( A \) is a Kripke consequence of \( \Gamma \)"") iff in each model \( \mathcal{X} \) such that if for all \( B \in \Gamma \mathcal{X} \vdash B \), then also \( \mathcal{X} \vdash A \). A is Kripke valid (K-valid) iff \( \emptyset \vdash A \) which will be written as \( \vdash A \). \[ \square \]

REMARK. Suppose \( \mathcal{X} (K, \leq, \vdash) \) is a Kripke model, \( k \in K \). The truncated model \( \mathcal{X}_k \) is \((K', \leq', \vdash')\) where \( K' = \{ k' : k' \geq k \}, \leq' \) is \( \leq \) to \( K' \) (the restriction of \( \leq \) to \( K' \)), and \( \vdash' \) is \( \vdash \) (the restriction of \( \vdash \) to \( K' \times \mathcal{P} \)).

It is easy to see that \( k \vdash A \) if \( \mathcal{X}_k \vdash' A \) if \( k \vdash' A \), since the definition of \( k \vdash A \) depends on \( \vdash \) for nodes \( k' \geq k \) only.

Note also that validity in \( \mathcal{X} \) is equivalent to validity at the bottom node of \( \mathcal{X} \), if there is one.
4. Lattices, Heyting algebras and complete Heyting algebras

4.1. Definition. A lattice is a partially ordered set (poset) \( A \) such that for each \( a, b \in A \) there is a least upper bound \( a \vee b \) (the join of \( a \) and \( b \)) and a greatest lower bound \( a \wedge b \) (the meet of \( a \) and \( b \)). This can be expressed by the following axioms: for all \( a, b, c \in A \):

\[
a \leq a \vee b, \quad b \leq a \vee b, \quad (a \leq c) \land (b \leq c) \rightarrow (a \vee b \leq c);
\]
\[
a \wedge b \leq a, \quad a \wedge b \leq b, \quad (c \leq a) \land (c \leq b) \rightarrow (c \leq a \wedge b).
\]

A zero or bottom element (denoted by 0 or \( \bot \)) in a lattice satisfies \( \forall a \in A (\bot \leq a) \), a top or unit element (denoted by 1 or \( \top \)) satisfies \( \forall a \in A (a \leq \top) \). If existing, top and bottom are unique. \( \Box \)

For the appropriate notions of sublattice and homomorphism we must regard a lattice as a structure \((A, \wedge, \vee)\). This suggests an alternative definition.

4.2. Definition. A lattice \((A, \wedge, \vee)\) is a set \( A \) with two binary operations \( \wedge, \vee \) such that for all \( a, b, c \in A \):

\[
L1 \quad a \wedge a = a, \quad a \vee a = a \quad \text{(idempotency)},
\]
\[
L2 \quad a \wedge b = b \wedge a, \quad a \vee b = b \vee a \quad \text{(commutativity)},
\]
\[
L3 \quad a \wedge (b \wedge c) = (a \wedge b) \wedge c, \quad a \vee (b \vee c) = (a \vee b) \vee c \quad \text{(associativity)},
\]
\[
L4 \quad a \vee (a \wedge b) = a, \quad a \wedge (a \vee b) = a \quad \text{(absorption)}.
\]

If, starting from this definition of a lattice, we put \( a \leq b := (a \wedge b = a) \) (or equivalently \( a \leq b := (a \vee b = b) \)) the two definitions of a lattice can be shown to be equivalent (exercise).

Two lattices \((A, \wedge, \vee)\) and \((A', \wedge', \vee')\) are said to be isomorphic if there is a bijection \( \varphi \) from \( A \) to \( A' \) which is a homomorphism with respect to \( \wedge, \vee \), i.e. \( \varphi(a \wedge b) = \varphi(a) \wedge' \varphi(b), \varphi(a \vee b) = \varphi(a) \vee' \varphi(b) \) for all \( a, b \in A \).

\((A', \wedge', \vee')\) is a sublattice of \((A, \wedge, \vee)\) if \( A' \subset A \), and \( \wedge', \vee' \) are the restrictions of \( \wedge, \vee \) to \( A' \). \( \Box \)
Remark. Sometimes a lattice is defined as having top and bottom; in this case one must add to L1–4 the axiom

\[ L5 \quad a \land 0 = 0, \quad a \lor 1 = 1. \]

4.3. Definition. A lattice \((A, \land, \lor)\) is **distributive** iff for all \(a, b, c \in A\):

\[ D1 \quad a \land (b \lor c) = (a \land b) \lor (a \land c), \]
\[ D2 \quad a \lor (b \land c) = (a \lor b) \land (a \lor c). \]

\(D1\) implies \(D2\) and vice versa (exercise). \(\Box\)

Examples. \(L_1, L_2\) are not distributive; \(L_3, L_4\) are distributive. See fig. 13.2.

We note in passing the following nice characterization of distributive lattices: a lattice is distributive iff it does not contain a sublattice isomorphic to \(L_1\) or \(L_2\) (proof e.g. in Birkhoff 1948, ch. 9).

![Diagrams of lattices](image)

Fig. 13.2.

4.4. Definition. Let \((A, \land, \lor)\) be a lattice, \(B \subseteq A\). \(a \in A\) is the **join** of \(B\) (notation: \(\lor B\)) iff \(a\) is a least upper bound of \(B\), i.e.

\[ \forall b \in B (b \leq a), \quad \forall c \in A (\forall b \in B (b \leq c) \rightarrow a \leq c). \]

Similarly, \(a \in A\) is the **meet** of \(B\) (notation: \(\land B\)) iff \(a\) is the greatest lower bound of \(B\). In particular, if \(\top\), respectively \(\bot\) exist, then

\[ \land \emptyset = \top, \quad \lor \emptyset = \bot. \]
If \( q(a) \) is an expression denoting a lattice element for each \( a \in A \), we sometimes use
\[
\bigwedge_{a \in A} q(a), \quad \bigvee_{a \in A} q(a)
\]
as alternative notations for \( \bigwedge \{ q(a) : a \in A \} \), \( \bigvee \{ q(a) : a \in A \} \), respectively. \( \Box \)

4.5. Definition. A lattice \((A, \land, \lor)\) is join-complete (meet-complete) iff joins (meets) exist for all \( B \subseteq A \). The lattice is complete iff it is join- and meet-complete. \( \Box \)

Example. The opens of any topological space \( T = (X, \emptyset) \) form a complete, distributive lattice with
\[
\begin{align*}
U \land U' &:= U \cap U', \\
U \lor U' &:= U \cup U', \\
\forall \forall' &:= \bigcup \{ V : V \in \forall' \}, \\
\land \forall' &:= \bigcap \{ V : V \in \forall' \}.
\end{align*}
\]

4.6. Proposition. A lattice is join-complete iff it is meet-complete.

Proof. Suppose \((A, \land, \lor)\) to be join-complete, let \( B \subseteq A \) and let \( B^* \) be the set of lower bounds of \( B \), i.e. \( B^* := \{ a \in A : \forall b \in B (a \leq b) \} \); put \( \land B := \lor B^* \). We leave it to the reader to verify that \( \land B \) indeed is the meet of \( B \). \( \Box \)

4.7. Definition. A Heyting algebra (Ha for short) is a structure \((A, \land, \lor, \bot, \rightarrow)\) such that \((A, \land, \lor)\) is a lattice with bottom \( \bot \), and \( \rightarrow \) a binary operation on \( A \) such that
\[
(a \land b \leq c) \iff (a \leq b \rightarrow c).
\]

We permit ourselves an abuse of language by using \( \rightarrow \) also for this lattice operation (because of its narrow relationship with logical implication) but this will not cause confusion.

Heyting algebra's are sometimes called pseudo-Boolean algebras. In the literature the lattice-operation \( \rightarrow \) is called implication or relative pseudo-complementation \((a \rightarrow b \) being the pseudo-complement of \( a \) relative to \( b \); \( a \rightarrow \bot \) is then the pseudo-complement of \( a \)). We write
\[
\neg a := a \rightarrow \bot. \Box
\]
4.8. Proposition.
(i) A Heyting algebra is a distributive lattice.
(ii) If \((A, \land, \lor, \perp, \to)\) is an Ha, and \(\lor B\) for some \(B \subset A\) exists, then for all \(a \in A\) the infinitary distributive law
\[
D: a \land \lor B = \lor \{a \land b : b \in B\}
\]
holds; in particular, the right-hand side of this equation is well-defined for all \(a \in A\).
(iii) Any complete lattice \((A, \land, \lor)\) satisfying D can be turned into an Ha by defining
\[
a \to b := \lor \{c : c \land a \leq b\}.
\]

Proof. Left as an exercise. □

4.9. Definition. A complete Heyting algebra (cHa for short) is a structure \((A, \land, \lor, \perp, \to, \land, \lor)\) such that \((A, \land, \lor, \perp, \to)\) is an Ha, and which is complete as a lattice with infinite joins and meets \(\land, \lor\). □

Of course, \(\land, \lor, \perp\) are in fact redundant, i.e., we may think of a cHa as a structure \((A, \land, \lor, \to)\). (As an object, a cHa may also be given as \((A, \land, \lor, \top)\); this gives rise to a different notion of homomorphism. cHa's regarded as structures with \(\land, \lor, \top\) as primitives are often called frames.)

4.10. Example. The opens of a topological space \(T = (X, \emptyset)\) form a cHa with \(\lor, \land, \land, \lor\) defined as before, and
\[
U \to V := \lor \{W : U \cap W \subset V\}.
\]
It can be shown that
\[
U \to V = \text{Int}\{x : x \in U \to x \in V\}
\]
(the \(\to\) on the right-hand side is a logical implication !), and classically also
\[
U \to V = \text{Int}(V \cup (X \setminus U)).
\]
We now turn to the notions of filter and ideal.

4.11. Definition. Let \((A, \land, \lor)\) be a lattice. An inhabited set \(F \subset A\) is a filter iff:
(i) \(a, b \in F \Rightarrow a \land b \in F\) (closure under \(\land\)),
(ii) \(a \in F\) and \(a \leq b \Rightarrow b \in F\) (upward closure).
These two conditions may be combined into
\[ a \land b \in F \iff a \in F \text{ and } b \in F. \]

The dual notion is that of an ideal; an inhabited set \( I \) is an ideal iff
\[ a \lor b \in I \iff a \in I \text{ and } b \in I, \]
or equivalently, expressed in two conditions

(iii) \( a \in I \) and \( b \in I \) \( \Rightarrow \) \( a \lor b \in I \),
(iv) \( a \in I \) and \( b \leq a \) \( \Rightarrow \) \( b \in I \).

A filter (ideal) is said to be proper if it is a proper subset of the lattice. A filter \( F \) is prime iff
\[ a \lor b \in F \Rightarrow a \in F \text{ or } b \in F. \]

The filter generated by a set \( X \) is the least filter \( F \supseteq X \) (this exists, since the intersection of a family of filters is again a filter). □

REMARK. The filter generated by \( X \) consists of all \( b \) such that \( b \geq a_1 \land \cdots \land a_n \) for some finite subset \( \{a_1, \ldots, a_n\} \subseteq X \). Equivalently, this filter can be described as the least set \( Y \) closed under \( a \in X \Rightarrow a \in Y; x, y \in Y \Rightarrow x \land y \in Y; x \in Y \land x \leq y \Rightarrow y \in Y. \)

4.12. Proposition. Let \( F \) be a filter, \( a \rightarrow b \notin F \). Then the filter generated by \( F \cup \{a\} \) does not contain \( b \).

Proof. Suppose \( b \) to be an element of the filter generated by \( F \cup \{a\} \); then by the preceding remark, \( b \geq c \land a \) for some \( c \in F \), hence \( c \leq a \rightarrow b \), but then \( a \rightarrow b \in F \) would follow. □

The next proposition is classical and requires Zorn’s lemma.

4.13. Proposition. Let \( F \) be a filter, \( a \notin F \). Then there is a prime filter \( F' \supseteq F, a \notin F' \). (Which would be an ultrafilter.)

Proof. The proof uses PEM and Zorn’s lemma (exercise). □
It is lengthy, but routine to check that $\Theta'$ must be a homomorphic image of $\Theta$; e.g. $6 \rightarrow 3 = 8$ in $\Theta'$ corresponds to $\vdash (A_6 \rightarrow A_3) \leftrightarrow A_8$, etc. On the other hand we can see at one stroke that $\not\models A_n^m$ for all $n \in \mathbb{N}$, since for the canonical valuation given by $[P] = 1$ we have, by the preceding remark, $[A_n^m] = n$. Accordingly none of the $A_n^m$ is provable, nor can any two $A_n^m, A_m^m$ with $n \neq m$ be equivalent; so $\Theta$ is isomorphic to $\Theta'$. □

5.6. Relationship between Kripke models and Ha-models. In one direction the correspondence is very easy. Let us call an Ha-model for an Ha $\Theta$ which consists of the opens of a topological space a topological model.

Every propositional Kripke-model $X = (K, \leq, \vdash)$ can be transformed into a topological model over $\Theta = (\Omega_K, \cap, \cup, \emptyset, \rightarrow)$, where $\Omega_K$ consists of all upwards monotone subsets of $(K, \leq)$; i.e. $X \in \Omega_K$ iff $X \subseteq K$ and $\forall k \in X (k \leq k' \Rightarrow k' \in X)$. $\Omega_K$ is a topology on $K$. The valuation $[\ ]$ determined by $X$ is

$[P] = \{ k : k \vdash P \}$.

We leave it as an exercise to the reader to show that for all $A$

$[A] = \{ k : k \vdash A \}$.

We observe the following.

Corollary (to 2.6.11). IPC is complete with respect to valuations in finite Ha.'s. □

For the converse, we have to argue classically.

5.7. Theorem (PEM, Zorn's lemma). Suppose a valuation in an Ha $\Theta = (A, \land, \lor, \bot, \rightarrow)$ to be given. We construct a Kripke model $X = (Pr(\Theta), C, \vdash)$, where $Pr(\Theta)$ is the set of (proper) prime filters of $\Theta$, and where $\vdash$ is given by

$F \vdash P : = [P] \in F$.

Under the correspondence just described, for all formulas $A$ of IPC

$F \vdash A \Leftrightarrow [A] \in F$.

Proof. By induction on the complexity of $A$. The only interesting case is implication. So let $A = B \rightarrow C$. If $[B \rightarrow C] \in F$, then for all $F' \supset F$ such that $[B] \in F'$, also $[B] \land ([B] \rightarrow [C]) \in F'$, so $[C] \in F'$. Hence (induction hypothesis) $\forall F' \supset F (F' \vdash B \Rightarrow F' \vdash C)$, i.e. $F \vdash B \rightarrow C$. If $[B \rightarrow C] \notin F$, then there is a prime $F' \supset F \cup ([B])$ with $[C] \notin F'$; so $F' \vdash B$ and $F' \not\models C$, hence $F \not\models B \rightarrow C$. The other cases are left to the reader. □