Logic Based Approach to AI

Michael Gelfond

Department of Computer Science

University of Texas at El Paso

El Paso, TX 79968

mgelfond@cs.ep.utexas.edu

Introduction

• **To make machines smart we need to teach them how to reason and how to learn.**

Most teaching is done by instructions.

Most learning comes from instructions.

We need efficient means of communication!

• **Languages differ according to the type of information their designers want to communicate to computers. Two basic types:**

ALGORITHMIC languages describe sequences of actions for a computer to perform.

DECLARATIVE languages describe properties of objects in a given domain.

First-order Languages

- **Developed in the beginning of the century (Boole, Frege, Peano,...).**

Original domain of applicability - math.

- **Statements constructed from atoms, e.g.** $p(\bar{t})$**, by applying logical connectives and quantifiers,e.g.,**

$\exists\, X \,\forall\, Y \; follows(Y, X)$ $\qquad$ $\forall\, X \; (p(X) \vee \neg p(X))$

- **Truth of atomic sentences depends on interpretation of symbols.**

Truth of compound sentences is defined through the truth of atomic sentences and the meaning of connectives.

Entailment and Inference

- **Set T of formulas ENTAILS formula F_n ($T \models F_n$) if F_n is true under every interpretation of symbols which makes T true.**

- **Relation $T \models F_n$ is established by giving a correct logical argument: a sequence of statements $F_0, \ldots, F_n$ where each F_i is an axiom or is obtained from previous statements by inference rules, e.g.**

$$\frac{A, \quad B \; if \; A}{B}$$

- **Almost all mathematical knowledge can be expressed in a simple first-order language.**

- **Few inference rules and a simple set of axioms are (in principle) enough to prove practically all mathematical theorems in our textbooks.**

Automating Inference

Establishing truth of F in T can be reduced to SEARCH in the space of all formal arguments (proofs).

PROBLEM - make search efficient

Three approaches: find

1. **efficient inference rules**
2. **efficient ways to apply these rules**
3. **classes of theories with comparatively simple proofs**

Study of deductive systems started 50 years ago (Herbrand, Gentzen, Pravitz,...), Robinson and Maslov - Resolution Rule (1965), various resolution strategies.

Pure Prolog

PROLOG - programming in logic (Kowalski, Colmerauer, Warren 1978)

Program is a collection of statements of the form

$$A_1(X) \wedge \ldots \wedge A_n(X) \supset A_0$$

Efficient query answering procedure implements the resolution inference rule and a simple, but efficient strategy of applying this rule.

In ten years the domain of applicability of predicate calculus expanded from mathematics to include natural language processing, databases, and even writing of compilers and operating systems.

Example

Family:

father(john,sam).

mother(mary,sam).

parent(X,Y) if father(X,Y).

parent(X,Y) if mother(X,Y).

child(X,Y) if parent(Y,X).

A reasoner, who "knows" this information about family should be able to answer queries:

Is Sam a child of John?

$?child(sam, john) \quad yes$

Who are Sam's parents?

$?parent(X, sam) \quad X = john$

Normative Statements

Some classes of statements of natural language are not easily translated into the language of predicate calculus.

1. NORMATIVE statements, i.e. statements including words "Normally, typically, as a rule", e.g. "normally, parents care about their children."

Statements of this sort usually accompanied by some exceptions: "John is an exception to this rule. He does not care for his son."

Normative statements are sometimes called DEFAULTS. A large part of our education seems to consists of learning various defaults, their exceptions, and the skill of reasoning with them.

Other Problematic Statements

2. Epistemic statements - statements about knowledge of a reasoning agent, e.g., "If you do not know if statement S is true, ask." "A person is assumed innocent unless proven guilty".

3. Counterfactuals - statements like "If Mozart had lived until 1990, he would have become the queen of England".

Non of the three types of statements mentioned above occur in the language of mathematics. We do not know if usual "classical" logical connectives can be used to CONVENIENTLY represent these types of knowledge.

Logic for commonsense reasoning

We need a theory of reasoning capable of dealing with defaults, epistemic statements, etc.

Unlike mathematics, the knowledge used by a "commonsense" reasoner will be INCOMPLETE and FREQUENTLY UPDATED.

Reasoning with such knowledge seems to be NONMONOTONIC, i.e. new information may force us to withdraw previous conclusions.

MY RESEARCH INTERESTS LIE IN THE DEVELOPMENT OF SUCH A THEORY.

DEFAULT REASONING.

Approaches:

1. **find a new language**
2. **define new entailment relation**
3. **generalize the notion of inference rule**
4. **do empirical study of reasoning in specific domains outside mathematics**

Of course, it would be nice to preserve as much from "classical" logic as possible.

Example

"NORMALLY, parents care about their children. John is an exception to this rule. He does not care about his son."

Default is expressed by the rule:

1. cares(X,Y) if parent(X,Y), NOT ab(1,X).

NOT is a NEW LOGICAL CONNECTIVE,

NOT P is true if "there is no evidence for P in the knowledge base"

Exceptions are expressed as

2. ab(1,john).

3. ¬ cares(john,sam).

Example (continued)

father(john,sam).

mother(mary,sam).

parent(X,Y) if father(X,Y).

parent(X,Y) if mother(X,Y).

child(X,Y) if parent(Y,X).

cares(X,Y) if parent(X,Y), NOT ab(X).

¬ cares(john,sam).

ab(john).

?cares(mary,sam) YES

?cares(john,sam) NO

?cares(sam,sam) UNKNOWN

NOT in Prolog

NOT appeared in Prolog as a purely technical devise. Its importance was recognized by Clark and Reiter in 1978. They gave first precise but not fully adequate semantics for it.

A better approach was found by Apt, Blair and Walker in 1985. Worked only for a special class of programs.

Different approach based on developments in AI - Gelfond and Lifschitz, 1987. Facilitated the flow of ideas between logic programming, databases and AI. In the 90's the language was extended by "classical" negation and several other new connectives.

Declarative logic programs

are sets of rules of the form

$$L_0 \; or \; \ldots \; or \; L_k \leftarrow L_{k+1}, \ldots, L_m, \; not \; \; L_{m+1}, \ldots, \; not \; \; L_n$$

where each L_i is a literal. The rule can be read as: "If $L_{k+1}, \ldots, L_m$ are believed and it is not true that $L_{m+1}, \ldots, L_n$ are believed then at least one of $L_0, \ldots, L_k$ must be believed".

- **A program determines a collection of ANSWER SETS – sets of ground literals representing possible beliefs of the program.**
- **A ground literal L is ENTAILED by a program Π ($\Pi \models L$) if L belongs to all of Π's answer sets.**
- **Answer to a query L is YES if $\Pi \models L$, NO if $\Pi \models \neg L$ and UNKNOWN otherwise.**

Examples

Program	Answer sets
$p(a) \leftarrow \neg p(b)$	$\{\ \}$

Program	Answer sets
$p(a) \leftarrow \neg p(b)$	$\{\neg p(b),\ p(a)\}$
$\neg p(b)$	

Program	Answer sets
$p(a) \leftarrow \neg p(b)$	$\{\neg p(b),\ p(a)\}$
$\neg p(b) \leftarrow\ not\ \ p(b)$	

Inference to compute $\models$

- **Need extra inference rules depending on the program, e.g.**

$$\frac{q \notin head(\Pi)}{not\ \ q}$$

- **THEOREM. There are programs with unenumerable sets of conclusions.**

- **SLDNF resolution and other similar procedures build in Prolog interpreters and in deductive databases inference engines can be viewed as a search for a proof in deductive system associated with a program.**

- **THEOREM. SLDNF resolution is sound w.r.t. answer set entailment.**

Directions of research

1. **Developing inference systems for declarative logic programs.**
2. **Developing software engineering methodology for logic programming paradigm.**
3. **Understanding mathematical properties of logic programs.**
4. **Formalizing particular domains and reasoning patterns.**
5. **Understanding relations with other non-monotonic systems.**

Applications: Null values in databases

Motivation: Investigate applicability of Declarative LP to representing incomplete knowledge in databases.

In database technology such info. is normally represented by NULL values. Problems:

- **How to define answer to a query?**

Relational algebra operations do not extend to relations with null values.

Translation to predicate calculus can't deal with Closed World Assumption: "Normally, if things are not in the tables they are false."

- **How to compute the answers?**

Proposed Solution

- **Use declarative LP to**

  (a) rewrite database tables

  (b) express the closed world assumption.

- **Use answer set semantics to define answer to a query.**

If Π is a program representing database tables and CWA then answer to a query $Q(X)$ is $\{t : \Pi \models Q(t)\}$.

Example: Complete database

The following table represents COMPLETE info about the summer schedule of the department:

Professor	Course
mike	pascal
john	c

Program representing this info:

1 $t(mike, pascal).$

2. $t(john, c).$

3. $\neg t(X, Y) \leftarrow \ not \ \ t(X, Y)$

? $t(mike, c)$ $\quad NO$

Incomplete Database 1

Professor	Course
mike	pascal
john	c
staff	lisp

STAFF is a null value which stands for an unknown professor (possibly different from Mike and John).

How to represent this information?

1 $t(mike, pascal).\quad t(john, c).\quad t(staff, lisp).$

2a. $\neg t(X, Y) \leftarrow not\ \ t(X, Y),\ not\ \ ab(X, Y)$

2b. $ab(X, Y) \leftarrow t(staff, Y)$

?$t(mike, c)\ NO,\quad$ **?**$t(mike, lisp)\ Unknown$

Incomplete Database 2

Professor	**Course**
mike	**pascal**
john	**c**
{mike,john}	**prolog**
staff	**lisp**

where {mike,john} represents the second type of nulls – "value unknown but one of the finite set of values".

1 $t(mike, pascal).\quad t(john, c).\quad t(staff, lisp).$

2. $t(mike, prolog)\ or\ t(john, prolog) \leftarrow$

3a. $\neg t(X, Y) \leftarrow\ not\ \ t(X, Y),\ not\ \ ab(X, Y)$

3b. $ab(X, Y) \leftarrow t(staff, Y)$

?$t(mike, c)\ NO,\quad$ **?**$t(mike, prolog)\ Unknown$

Next Step

- **The above translations can be generalized to give a general definition of answer to a query for databases with various types of null values. Some relations with other approaches have been established. Next step is to**

- **Check if "general purpose" LP methods can (in principle) be used to answer queries.**

A query-answering algorithm sound w.r.t. answer set semantics is implemented based on SLG system by Warren and Chen. Students are working on combining it with standard database systems.

Applications to reasoning about actions

Domain has three sorts of objects:

situations $S, S', \ldots$;

fluents $F, F', \ldots$;

actions $A, A', \ldots$

Language should be able to represent

1. **Facts about values of fluents and occurrences of actions in particular situations.**
2. **Effects of actions**
3. **Inertia axiom: "Things normally tend to stay the same."**

Example: Jack's trip to the airport

$initial\ facts:$

$(f1)$ $home$ **at** s_0.

$(f2)$ $\neg at\_airport$ **at** s_0.

$(f3)$ has_car **at** s_0.

$Laws:$

$(l1)$ $rent$ **causes** has_car **if** $true$.

$(l2)$ hit **causes** $\neg has\_car$ **if** $true$.

$(l3)$ $drive$ **causes** $at\_airport$ **if** has_car.

$(l4)$ $drive$ **causes** $\neg home$ **if** has_car.

$(l5)$ $pack$ **causes** $packed$ **if** $home$.

} D

Goal: bring packed suitcase to the airport

Domain independent axioms

(f after α at S) is read as "assuming that the sequence α of actions occur in situation S a fluent f will be true in the resulting situation."

[] is an empty sequence of actions.

EFFECT AXIOMS

$$(F \textbf{ after } [] \textbf{ at } S) \leftarrow (F \textbf{ at } S).$$

$$(F \textbf{ after } A \circ \alpha \textbf{ at } S) \leftarrow (A \textbf{ causes } F \textbf{ if } P]),$$
$$(P \textbf{ after } \alpha \textbf{ at } S).$$

If F is true after executing A in any situation in which P is true and P is true after executing α in situation S then F is true after executing $A \circ \alpha$ in S.

Inertia Axiom

Assuming that F will be true after execution of α at S, we can typically conclude that F will be true after the execution of $A \circ \alpha$.

$$(F \textbf{ after } A \circ \alpha \textbf{ at } S) \leftarrow (F \textbf{ after } \alpha \textbf{ at } S),$$
$$not\ ab(F, A, \alpha, S)$$

Inertia Axiom is not applicable to F, A, α, S if F is true after executing A in any situation in which P is true and P MAY BE true after execution of α in S.

$$ab(F, A, \alpha, S) \leftarrow (A \textbf{ causes } F \textbf{ if } P)$$
$$not\ (\neg P \textbf{ after } \alpha \textbf{ at } S),$$

Actual Events

To represent actual occurrences of events we use two predicates S_2 **immediately_follows** S_1 **and** A **occurs_at** S **and the following axioms:**

$(F$ **at** $S_2) \leftarrow (S_2$ **immediately_follows** $S_1)$,

$(A$ **occurs_at** $S_1)$,

$(F$ **after** A **at** $S_1)$

$\neg current(S_1) \leftarrow (S_2$ **immediately_follows** $)$

$current(S) \leftarrow not\ \neg current(S)$

$currently(F) \leftarrow current(S), F$ **at** S

Finding Plans

The simpliest way of finding plans is to generate "candidate" sequences of actions and use the above theory to check if the goal will be true after the execution of the sequence.

$$find\_plan(\alpha, G) \leftarrow \; generate(\alpha),$$
$$current(S),$$
$$G \textbf{ after } \alpha \textbf{ at } S.$$

Use of domain dependent heuristics (expressible in the language of LP) seems to allow efficient generation of "candidates".

Architecture for Intelligent Agents

The planner is incorporated into a more general architecture for intelligent agents represented by the following loop:

1. $observe$;

2. $select\_goal(G)$;

3. $find\_plan([a_1, \ldots . a_n], G)$;

4. $execute(a_1)$;

At stages (1) and (4) the theory D is expanded by new information about actual occurrences of events (both, caused by and observed by the agent).

Jack's problem revisited

1. $find\_plan(X, [packed, at\_airport])$;

   $X = [pack, drive]$

2. $execute(pack)$

   Expands D by ($pack$ **occurs_at** s_1) and

   (s_1 **immediately_follows** s_0)

3. $observe(hit)$

   Expands D by (hit **at** s_2) and

   (s_2 **immediately_follows** s_1)

(Notice, that $D \models currently(\neg has\_car)$ and hence the old plan is invalidated)

4. $find\_plan(X, [packed, at\_airport])$;

   $X = [rent, drive]$

Conclusions

- **DLP with answer set entailment relation provides powerful model of reasoning with incomplete information in a frequently changing world. It is capable of conveniently representing defaults and epistemic statements commonly used in various domains.**
- **Development of inference mechanisms approximating this entailment is based upon advances in LP and theorem proving. First powerful systems are becoming available for experimentation. Work started at applying them to specific domains.**
- **Experimentation and better understanding of mathematical properties of declarative logic programs is crucial for further development.**