COSC410 Logic for AI

Datalog

Richard A. O'Keefe

26 May 2016

Key Topics

- Unary predicate calculus
- Decidable fragments of first-order logic
- Horn clauses
- Pure Datalog and its semantics
- Adding arithmetic
- Negation and stratification

Unary Predicate Calculus

- terms are constants and variables
- there are only unary predicates
- except that $=$ may be allowed.
- usual connectives
- usual quantifiers

Löwenheim-Skolem 1915

- If ϕ is a sentence in unary predicate calculus that has a model, then it is true in some interpretation whose domain has at most $2^p.v$ members, where p is the number of predicate symbols in ϕ and v the number of variables.
- This basically works because there are no binary predicates to mix up two different variables.
- There are only finitely many such interpretations (up to isomorphism), so exhaustive search is possible.

Why does this matter?

- UPC is perfectly suited to expressing *type hierarchies*.
- Every chess piece is black or white but not both. Every chess piece is a pawn, a rook, a knight, a bishop, a king, or a queen, and cannot be any two of these. Some birds are fliers. Some fish are fliers. Some snakes are fliers. Not all fliers are breathers. And so on.
- We'll see this kind of stuff in the last lecture.

More generally

- Even unifiability is undecidable in 2nd-order logic
- 1st-order predicate calculus is semi-decidable
- propositional calculus is decidable
- **These are not the only possible logics!**
- Since UPC is decidable, it makes sense to look for other decidable fragments of 1st-order logic.

Kowalski form for clauses

- Given a clause $\{P_1, \dots, P_m, \neg N_1, \dots, \neg N_n\}$,
- it's an implication, so write it as
- $P_1, \dots, P_m \leftarrow N_1, \dots, N_n$.
- Since ASCII lacks $\leftarrow$, use :-.
- black(X), white(X) :- chess_piece(X).
- On the left, the *head*, on the right, the *body*.
- Write empty body as true.

Horn clauses

- Named for Alfred Horn, 1951.
- Any clause with *at most one positive literal*.
- With exactly one positive literal, called a *definite* clause.
- If $m = 1$ and $n = 0$, sometimes called a *fact*.
- If $m = 1$ and $n > 0$, sometimes called a *rule*.
- If $m = 0$, called a *goal* or *query*.

A model result

Every set of definite clauses Π has a *unique* minimal model M. An atomic formula A is implied by Π if and only if A is true in M.
Kowalski and van Emden, 1976.

Are we there yet?

- A set of *propositional* Horn clauses can be solved in linear time.
- A set of *first-order* Horn clauses is still undecidable because that's a Turing-complete programming language called (pure) Prolog.
- The proof is by giving a simple method of converting any Turing machine to a set of definite clauses and an initial state to a query; finding out whether the query is satisfiable is as hard as running the machine.

Following the lead of unary predicate calculus

A **Datalog** database is a collection of definite clauses where

- Terms are just constants and variables, there are no function symbols with arity > 0.
- Every variable that occurs in the head must occur in the body.

Remember this for the exam.

Example

```
father(F, C) :- parent(F, C), male(F).
mother(M, C) :- parent(M, C), female(C).
child(C, P) :- parent(P, C).
son(C, P) :- child(C, P), male(C).
daughter(C, P) :- child(C, P), female(C).
ancestor(A, D) :- parent(A, D).
ancestor(A, D) :- parent(A, P), ancestor(P, D).
```

Databases

- Ignoring NULL, a *table* in a relational database is just a collection of atomic sentences about individuals. That is, it's a collection of facts.
- In Datalog, the *extensional data base* is a set of facts. That is, clauses with an empty body. A clause with an empty body cannot have variables in the head.
- The *intensional data base* is a set of rules to be used to infer additional facts. Think of it as defining views, except that the rules can be recursive.

One-step inference

$T \uparrow (e) = \{h | (h \leftarrow b_1, \ldots, b_n)$ is an instance of a rule in the intensional data base
$\wedge b_1 \in e \wedge \cdots \wedge b_n \in e\} \cup e$.
That is, for each rule, for each way of matching the elements of the body with facts currently known, add the corresponding instance of the head to the new facts. This can be done in $O(r.|e|^n)$ time where n is the number of atoms in the longest body and r is the number of rules.

Complete inference

- $T \uparrow^0 (e) = e$
- $T \uparrow^{k+1} (e) = T \uparrow (T \uparrow^k (e))$
- $T \uparrow^\omega (e) = \lim_{k \to \infty} T \uparrow^k (e)$

This is the general semantics for pure Horn clause programs. For Datalog, there is always a *finite* k such that $T \uparrow^\omega (e) = T \uparrow^k (e)$ because there are only finitely many possible facts.

Forward *vs* backward chaining

- Forward chaining: when you find out the body is true, conclude the head. This preserves *truth*.
- Backward chaining: to try to prove an instance of the head, try to prove each element of the body. This preserves *relevance* to our goal.
- Logic programming languages use backward chaining.
- For data bases, we don't want to prove useless facts *or* waste time exploring things that can't be true.

Enriching the language: arithmetic

We can allow arithmetic definitions (*Var* is *expr*) and comparisons ($e_1 \{<, \geq, >, \leq, =, \neq\} e_2$) in the body of a rule as long as

- definitions are evaluated only after all the variables in the *expr* have been bound by an ordinary literal or another definition, and
- comparisons are evaluated only after all the variables in their expressions have been bound by ordinary literals or by definitions.

That is, there has to be a partial order on the literals in the body of a rule that uses *data flow* to ensure that arithmetic never encounters a variable that doesn't have a value yet.

Enriching the language: negation

- Considered as logic, putting a negation in the body of a rule is equivalent to adding a disjunction in the head.
- But the semantics we've given doesn't handle disjunctions in the head.
- We don't want the Barber paradox in our data base, so we don't want any recursive negations. And we don't want negations to solve for variables.

Banishing barbers: stratification

A program is *stratified* iff the set of predicate symbols is divided into *strata* S_0,S_1, $\ldots S_z$ such that

- S_0 contains the extensional predicate symbols
- S_{i+1} contains intensional predicate symbols defined by rules in which the plain atoms mention predicate symbols in $\bigcup_{j=0}^{i+1} S_j$ and the negated atoms mention predicate symbols in $\bigcup_{j=0}^{i} S_j$ (*i.e.*, not in S_{i+1}).

This means there can be no negation loops.
Remember the constraints for negation too.

Banishing barbers: binding

Just as arithmetic comparisons are used as filters to reject combinations found by joining positive atoms, so negated atoms are used as filters. The data flow partial order must be extended to include negated atoms: a negative atom may not be checked until after all of the variables in it have been filled in by positive atoms or arithmetic definitions.

Stratified semantics

- define $T_i \uparrow (e)$ to be the one step inferences using rules for predicates in stratum S_i.
- $M_0(e) = e$
- $M_{i+1}(e) = T_{i+1} \uparrow^{\omega} (M_i(e))$

That is, we start with the given facts, and then derive all the consequences from each stratum, one at a time.

A semantics is not an interpreter

- All this stuff about $T \uparrow^{\omega}$ and M_z tells us that a Datalog program has a meaning, what the meaning is, and that the meaning can be derived in finite time.
- That is not how Datalog is actually implemented. That involves backwards chaining, tabling (= memoisation = caching results so they're not recomputed), incremental updates, Magic Sets, and other cleverness we lack time to cover.

Comparison with relational algebra

- Datalog can't do relational division.
- Relational algebra can't do recursion.
- Datalog is based on 2-valued logic.
- SQL is based on an inconsistent 3-valued logic.
- I find Datalog much easier to get correct.

Next week

We'll be looking at Description Logic and the Semantic Web. There are overlaps with this week's topic. . .