Lecture 10: Theorem Proving and Question Answering

Stephen.Pulman@comlab.ox.ac.uk

If we assume that some kind of logic is a good knowledge and semantic representation language, then we need to be able to write programs which can make inferences from sets of sentences of that logic. Such a program is usually called a 'theorem prover'.

- Incomplete but simple inference systems
- Skolemisation; normal forms; clausal form
- Horn Clauses and Prolog
- Resolution

S. Russell and P. Norvig 2003 Artificial Intelligence: A Modern Approach (Second Edition) Prentice Hall. (Chapter 9)
W. F. Clocksin and C. S. Mellish, Programming in Prolog, Springer Verlag, (3rd edition), 1987.
A. Bundy 1983 The Computer Modelling of Mathematical Reasoning, London: Academic Press
C-L. Chang and R C-T Lee 1983 Symbolic Logic and Mechanical Theorem Proving, NY: Academic Press.

BACKWARD CHAINING
given a set of rules of the form A $\rightarrow$ B, and a set of atomic facts $C_1 \ldots C_n$.
trying to prove P, if P = some C_i, then stop with success
trying to prove Q, given P $\rightarrow$ Q, try to prove P.

FORWARD CHAINING
trying to prove Q, if Q = some C_i, then stop with success
trying to prove Q, given P and P $\rightarrow$ R, add R and try to prove Q

No logical difference, and both can loop. Atomic = does not contain a logical connective.

First order unification: find values for (implicitly universally quantified) variables s.t. substituting values in makes the two expressions identical:

sExpresssion 1		Expression 2	Substitutions
like(john,mary)	=	like(X,Y)	{X=john,Y=mary}
like(X,mary)	=	like(john,Y)	{X=john,Y=mary}
like(john,mary)	=	like(X,X)	fail
like(john,mary)	=	like(X,bill)	fail
like(john,Y)	=	like(X,X)	{X=Y=john}
like(X,father-of(X))	=	like(Y,Y)	fail - 'occurs check'

Decidable, deterministic, almost linear complexity...

Matching of two expressions is by first order unification So finding the value of some variables is like answering a 'Wh' question:

1. man(X) $\rightarrow$ mortal(X)
2. man(socrates)

? mortal(socrates) via 1: man(socrates) unifies with 2: answer yes.

? mortal(Y) via 1: man(Y) unifies with 2: Y = socrates.

We can extend this to include conjunction and disjunction:

prove(P) if P unifies with an atomic fact.

prove(P $\wedge$ Q) if prove(P) and prove(Q).

prove(P $\vee$ Q) if prove(P) or prove(Q).

prove(Q) if there is a rule $\alpha \rightarrow$ Q, and prove(α), where α may be a combination of conjunctions and disjunctions.

A Prolog-like inference system

We can do quite a lot with a backward chaining system, by adding conjunction, disjunction, and also negation (by failure) to the antecedents of implications. We get something a bit like Prolog.

male(X) $\wedge$ parent(X,Y) $\rightarrow$ father(X,Y)
male(X) $\wedge$ parent(Y,X) $\rightarrow$ son(X,Y)
brother(X,Y) $\vee$ sister(X,Y) $\rightarrow$ sibling(X,Y)
male(john). male(bill). parent(bill,john).

However, 'negation by failure' (not(P) is true if P cannot be proved) does not work quite like classical negation:

farmer(john). rich(john). farmer(fred).

? farmer(X) $\wedge$ not(rich(X)) : X = fred

? not(rich(X)) $\wedge$ farmer(X) : false

But in classical first order logic these two expressions are logically equivalent.

Normal Forms

Prolog-like inference mechanisms require logical expressions to be in a particular 'normal form' known as Horn Clauses. In fact almost all theorem proving methods assume some kind of normal form which reduces the syntactic variety of logically equivalent expressions. The first step of turning an arbitrary FOPC expression into a normal form is usually 'skolemisation', which enables us to eliminate explicit quantifiers.

The rules for skolemisation are, for simple cases:

$\forall$x.P $\Rightarrow$ P - and any variables in P are interpreted as universally quantified.
$\exists$x.P $\Rightarrow$ P - with all occurrences of x in P replaced by a new constant like sk1, sk2 etc.

Examples:
$\forall$X.animal(X) = animal(X).
$\exists$X.animal(X) = animal(sk1)

These `sk1,sk2, ...` are new function symbols, here functions of no argument. When there is a mixture of existential and universal quantifiers an existential translates as a function whose arguments are the variables of the universal quantifiers having scope over it:
$\forall$X. $\exists$Y.likes(X,Y) =likes(X,sk3(X))
$\forall$A.$\exists$B.$\forall$C.$\exists$D.p(A,B,C,D) =p(A,sk4(A),C,sk5(A,C))

NEGATION makes the quantifiers behave like their duals:

$\neg\exists X.\text{tall}(X) \quad = \neg\text{tall}(X)$
$\neg\forall X.\text{tall}(X) \quad = \neg\text{tall}(\text{sk6})$

Antecedents of conditionals behave like negatives:

$(\exists X.\text{rich}(X)) \rightarrow (\exists Y.\text{happy}(Y))$

not: $\text{rich}(\text{sk7}) \rightarrow \text{happy}(\text{sk8})$
but: $\text{rich}(X) \rightarrow \text{happy}(\text{sk8})$

(consider if we had already asserted 'rich(john)').

Notice that in a Prolog-like inference system, we have to skolemise queries as if they were negatives:

assert	$\exists X.\text{p}(X)$	$= \text{p}(\text{sk9})$
query	$\exists X.\text{p}(X)$	$= \text{p}(X)$
assert	$\forall X.\text{p}(X)$	$= \text{p}(X)$
query	$\forall X.\text{p}(X)$	$= \text{p}(\text{sk10})$

CONJUNCTIVE NORMAL FORM

A *positive literal*: an expression which contains no connectives (i.e. just a predicate and its arguments). A *negative literal*: a positive literal preceded by a negation. A *clause*: a set of positive and/or negative literals, implicitly disjoined. A *Horn Clause*: a clause with at most one positive literal. A *definite clause*: a clause with exactly one positive literal.

Conjunctive normal form (CNF): a conjunction of clauses e.g.

$(p \vee q \vee \ldots) \wedge (x \vee y \vee \ldots) \wedge \ldots$

Disjunctive normal form is the other way round - a disjunction of implicitly conjoined literals. We can transform any FOPC expression into CNF by first skolemising, and then applying the following equivalences:

1. $P \rightarrow Q \equiv (\neg P) \vee Q$
2. $\neg\neg P \equiv P$
3. $\neg(P \vee Q) \equiv (\neg P) \wedge (\neg Q)$
4. $\neg(P \wedge Q) \equiv (\neg P) \vee (\neg Q)$
5. $P \vee (Q \wedge R) \equiv (P \vee Q) \wedge (P \vee R)$

1 removes implications, 3 and 4 move negations inwards, and 5 moves conjunctions to the top level. These equivalences (and others like $P \wedge Q \equiv Q \wedge P$) are applied recursively until no more can apply.

EXAMPLE

All the nice girls like a sailor

$\forall X.(\text{nice}(X) \wedge \text{girl}(X) \rightarrow (\exists Y.\ \text{sailor}(Y) \wedge \text{like}(X,Y)))$

Skolemise:

$\text{nice}(X) \wedge \text{girl}(X) \rightarrow \text{sailor}(\text{sk1}(X)) \wedge \text{like}(X,\text{sk1}(X)))$

Remove implications:

$\neg(\text{nice}(X) \wedge \text{girl}(X)) \vee \text{sailor}(\text{sk1}(X)) \wedge \text{like}(X,\text{sk1}(X)))$

Move negation in:

$(\neg\text{nice}(X) \vee \neg\text{girl}(X)) \vee \text{sailor}(\text{sk1}(X)) \wedge \text{like}(X,\text{sk1}(X)))$

Move conjunctions out:

$(\neg\text{nice}(X) \vee \neg\text{girl}(X)) \vee \text{sailor}(\text{sk1}(X))$
$\wedge$
$(\neg\text{nice}(X) \vee \neg\text{girl}(X)) \vee \text{like}(X,\text{sk1}(X))$

Remove unnecessary brackets to give CNF:

$(\neg\text{nice}(X) \vee \neg\text{girl}(X) \vee \text{sailor}(\text{sk1}(X))$
$\wedge$
$(\neg\text{nice}(X) \vee \neg\text{girl}(X) \vee \text{like}(X,\text{sk1}(X))$

Remove connectives to give clauses implicitly conjoined:

$\{\neg\text{nice}(X), \neg\text{girl}(X), \text{sailor}(\text{sk1}(X)\}$
$\{\neg\text{nice}(X), \neg\text{girl}(X), \text{like}(X,\text{sk1}(X)\}$

Now we can transform each conjunct separately (or leave it alone: this is called 'clausal form'). One transformation might be to 'implicational form'. First collect negative and positive literals:

$\neg P \vee \ldots \neg Q \vee A \vee \ldots. B$

- by associativity of $\vee = (\neg P \vee \ldots \neg Q) \vee (A \vee \ldots. B)$

by 4: $(\neg P \vee \ldots \neg Q) = \neg(P \wedge \ldots Q)$

by 1: $\neg(P \wedge \ldots Q) \vee (A \vee \ldots. B) = (P \wedge \ldots Q) \rightarrow (A \vee \ldots. B)$

If no negatives, implicational form = **true** $\rightarrow$ **P**, sometimes written '$\rightarrow$ **P**'; if no positives, implicational form = **P** $\rightarrow$ **false**, also written '**P** $\rightarrow$', where P can be a disjunction or conjunction respectively. NB:

true	$\rightarrow$	P		P	$\rightarrow$	false
T T	T F	T F		T F	F T	F F

true $\rightarrow$ **P** where P is a single literal = a Prolog unit clause. No Prolog equivalent for **P** $\rightarrow$ **false**, or for implicational forms with more than one disjunct in the consequent, like **P** $\rightarrow$ **Q** $\vee$ **R**. This means that Prolog, or any similar backward/forward chaining mechanism restricted to definite clauses, cannot deal properly with full disjunction or negation.

RESOLUTION

Resolution is an inference rule that is complete for FOPC. Given a form like:

GIVEN: P $\rightarrow$ R $\vee$ S, AND: R $\rightarrow$ Q
CONCLUDE: P $\rightarrow$ Q $\vee$ S

GIVEN: P $\wedge$ Q $\rightarrow$ R, AND: X $\rightarrow$ P $\vee$ Y
CONCLUDE: X $\wedge$ Q $\rightarrow$ Y $\vee$ R

The Ps and Rs that 'cancel out' in the resolving clauses need only unify, not be identical, and the set of substitutions resulting from unification is applied to the result of resolution.

To prove that some conclusion follows from a set of axioms we transform the axioms into implicational form, transform the **negation** of the conclusion into implicational form and apply the resolution rule until we end up with the 'empty clause' (or equivalent, depending on notation) signifying a contradiction. If the negation of the conclusion is inconsistent with the axioms then the positive version follows from them.

EXAMPLE

Every car has a steering wheel.

$\forall X.car(X) \rightarrow \exists Y.sw(Y) \wedge has(X,Y)$
A1: $car(X) \rightarrow sw(sk1(X))$
A2: $car(X) \rightarrow has(X,sk1(X))$

A Skoda is a car.

B: $\rightarrow car(skoda)$ (think of a clause '$\rightarrow$ P' as 'true $\rightarrow$ P')

Does a Skoda have a steering wheel?

$\exists Z.sw(Z) \wedge has(skoda,Z)$

C: (negated): $sw(Z) \wedge has(skoda,Z) \rightarrow$

(think of this as: $sw(Z) \wedge has(skoda,Z) \rightarrow$ false)

Resolve C with A1: substitutions = $\{Z/\ sk1(X)\}$

D: $car(X) \wedge has(skoda,sk1(X)) \rightarrow$

Resolve D with A2: substitutions = $\{X/skoda\}$

E: $car(skoda) \wedge car(skoda) \rightarrow$
E: $= car(skoda) \rightarrow$

Resolve E with B to give the empty clause.

It is easy to see that backward and forward chaining are just special cases of resolution, where the form of the resolvents is limited:

man(socrates)
man(X) $\rightarrow$ mortal(X)
———————————-
?mortal(socrates)

1. true $\rightarrow$ man(socrates)
2. man(X) $\rightarrow$ mortal(X)
3. mortal(socrates) $\rightarrow$ false (negate the query)

Resolve 2 with 3:

4. man(socrates) $\rightarrow$ false

Resolve 4 with 1:

5. true $\rightarrow$ false (contradiction).

Resolution is a general method but it is difficult to make it efficient for all types of problem.

GETTING ANSWERS

Frequently in doing a proof we are interested in the values of variables that are instantiated during the proof. But the simple resolution method does not retain these, since we just end up with the empty clause.

We can retrieve the values of variables by disjoining a dummy 'answer' literal to the (clausal form of the) theorem to be proved. Change the termination condition accordingly, because a clause consisting of just this literal will now count as the empty clause.

man(socrates); $\forall$X.man(X) $\rightarrow$ mortal(X)
Query: $\exists$Y.mortal(Y) ?
Clausal form with added disjunct: $\neg$mortal(X) $\vee$ answer(X)
Equivalent implication form: mortal(X) $\rightarrow$ answer(X)

1. $\rightarrow$ man(socrates)
2. man(X) $\rightarrow$ mortal(X)
3. mortal(Y) $\rightarrow$ answer(Y)

Resolve 2 with 3:

4. man(Y) $\rightarrow$ answer(Y)

Resolve 4 with 1:

5. $\rightarrow$ answer(socrates)

NL QUESTION ANSWERING

Now we can hook up our parser and semantic interpreter to a theorem prover to build a simple NL question answering system.

Parse the sentence and extract the logical form.

If it's a declarative, turn the LF into clausal/implicational forms and add them to the current axioms.

If it's a question, turn ¬LF into clausal/implicational forms, add them to the current axioms temporarily and try to derive the empty/answers clause.

We conclude with a demonstration of such a system....