Comp24112: Symbolic AI
Lecture 6: First-order logic and AI

Ian Pratt-Hartmann

Room KB2.38: email: ipratt@cs.man.ac.uk

2016–17
Outline

Logic and planning

The SATCHMO prover
A well-known paradigm case of how logic can be used to reason about a simple situation is the blocks world:

Initial situation:
- on(a, b)
- on(b, table)
- on(c, table)
- clear(a)
- clear(c)

Goal conditions:
- on(a, b)
- on(b, c)
- clear(a)
A natural predicate calculus description of the above blocks world might be:

\[
on(a, b) \land \text{clear}(a) \land \text{on}(b, \text{table}) \land \text{clear}(c) \land \text{on}(c, \text{table}).\]

However, we need to relativize such descriptions to the situations in which they hold, e.g. by means of a situation variable.

Letting \( s_0 \) denote the initial situation, we write:

\[
on(a, b, s_0) \land \text{clear}(a, s_0) \land \text{on}(b, \text{table}, s_0) \\
\quad \text{clear}(c, s_0) \land \text{on}(c, \text{table}, s_0).
\]

The predicate calculus, used with situation variables to reason about changing environments is called the situation calculus.
• In the present example, there is only one type of action: that of moving a block from one location to another.

• We shall write

\[ \text{move}(x, y, z) \]

\text{to denote the action of moving } x \text{ (a block) from } y \text{ (a block or the table) to } z \text{ (a block or the table).}

• To reason about the effects of actions, we introduce a function \( \text{do}(\alpha, s) \) where \( \alpha \) is an action and \( s \) a situation.

• E.g.

\[ \text{do}(\text{move}(a, c, \text{table}), s_0) \]

denotes the situation resulting from moving \( a \) from \( c \) to the table in situation \( s_0 \).

• Thus, we name situations by means of the actions that produce them.
• Consider the action-type move($x, y, z$) of picking up a block $x$ from one resting place $y$ (another block or the table), and putting it on $z$ (again, another block or the table).

• This action can be performed if $x$ is on $y$, if $x$ is clear (has nothing resting on it), and if $z$ is clear.

• If the action is performed, then $x$ will be on $z$, $y$ will be clear
• Expressed in predicate calculus:

$$\forall x \forall y \forall z \forall s((\text{on}(x, y, s) \land \text{clear}(x, s) \land \text{clear}(z, s) \land x \neq z) \rightarrow$$
$$\text{on}(x, z, \text{do}(\text{move}(x, y, z), s)) \land$$
$$\text{clear}(y, \text{do}(\text{move}(x, y, z), s)))$$

• In words: for any $x$, $y$, $z$, $s$, if $x$ is on $y$ and $x$ and $z$ are clear in situation $s$, then $x$ is on $z$ and $y$ is clear in the situation that results by moving $x$ from $y$ to $z$ in situation $s$. 
• In addition, we might adopt an axiom defining the predicate clear\((x, s)\)

\[\forall x \forall s(\text{clear}(x, s) \iff (\neg \exists y \text{on}(y, x, s) \lor x = \text{table})).\]

• In words: for all \(x\) and \(s\), \(x\) is clear in situation \(s\) if and only if it is not the case that there exists a \(y\) such that \(y\) is on \(x\) in situation \(s\), or \(x\) is the table.

• And we need axioms stating that the blocks are distinct!

\[a \neq b \quad a \neq c \quad \ldots\]
If actions are represented in this way, we can use theorem-proving techniques to verify that a plan will achieve a given set of goals.

- Consider the plan:
  
  ```
  move(a,b,table)
  move(b,table,c)
  move(a,table,b)
  ```

- The situation that results from executing this plan in $s_0$ will be
  
  ```
  do(move(a, table, b),
      do(move(b, table, c),
          do(move(a, b, table), $s_0$)))
  ```

- We can verify that the plan will work if we can prove that the goal conditions obtain in this situation.
• Question: Suppose we take as premises a description of the initial situation plus the planning axiom plus the axiom defining $\text{clear}(x, s)$ in terms of $\text{on}(x, y, s)$. Can we prove the following formula?

$$\text{on}(a, b, \Delta) \land \text{on}(b, c, \Delta) \land \text{clear}(a, \Delta)$$

where $\Delta$ is the expression

$$\text{do}(\text{move}(a, \text{table}, b),$$
$$\text{do}(\text{move}(b, \text{table}, c),$$
$$\text{do}(\text{move}(a, \text{b, table}), s_0)))) ?$$

• The answer is no!
• What we need are axioms like these:

\[\forall x \forall y \forall z \forall u \forall v \forall s ((\text{on}(x, y, s) \land \text{clear}(x, s)) \land \\
\text{clear}(z, s) \land x \neq z \land u \neq y \land \text{on}(v, u, s)) \rightarrow \\
\text{on}(v, u, \text{do}(\text{move}(x, y, z), s)))\]

\[\forall x \forall y \forall z \forall w \forall s ((\text{on}(x, y, s) \land \text{clear}(x, s)) \land \\
\text{clear}(z, s) \land x \neq z \land w \neq z \land \text{clear}(w, s)) \rightarrow \\
\text{clear}(w, \text{do}(\text{move}(x, y, z), s)))\]
Now, and only now, can we prove the critical theorem:

\[\text{on}(a, b, \Delta) \& \ \text{on}(b, c, \Delta) \& \ \text{clear}(a, \Delta)\]

where \(\Delta\) is the expression

\[
\text{do}(\text{move}(a, \text{table}, b), \\
\quad \text{do}(\text{move}(b, \text{table}, c), \\
\quad \quad \text{do}(\text{move}(a, b, \text{table}), s_0)))
\]

- Predicate calculus buffs might like to try it
- Prolog hackers might like to write a theorem-prover to do it.
• Of course, *verifying* that a given plan will work is one thing, *finding* the plan another.

• But in fact, the task of finding a plan can be treated as a theorem-proving problem.

• If there is a plan which will achieve the goal conditions, then the following formula can be proved (given our axioms):

\[ \exists s (\text{on}(a, b, s) \& \text{on}(b, c, s) \& \text{clear}(a, s)) \]

• What we need is a theorem-prover that *finds a value* for \(s\).
Outline

Logic and planning

The SATCHMO prover
• The SATCHMO theorem prover (Manthey and Bry, 1989)


• This simple program works on clauses of the form:

\[
a_1 \land \ldots \land a_m \rightarrow b_1 \lor \ldots \lor b_n
\]

\[
\text{true} \rightarrow b_1 \lor \ldots \lor b_n
\]

\[
a_1 \land \ldots a_m\land \rightarrow \text{false}
\]

where the \(a_i\) and \(b_j\) are positive literals

• Any clause

\[
c_1 \lor \ldots \lor c_n
\]

can be put into this form by putting the negative literals on the left of the \(\rightarrow\) and the positive literals on the right.
- Preliminary version of SATCHMO:

```prolog
:- op(1050, xfy, --->).

satisfiable:-
  is_violated(C), !,
  satisfy(C),
  satisfiable.

satisfiable.

component(X, (Y;Z)):-
  !,
  (X=Y; component(X,Z)).

component(X,X).

on_backtracking(_X).

on_backtracking(X):-
  call(X), !, fail.

is_violated(C):-
  (A ---> C),
  call(A),
  not(C).

satisfy(C):-
  component(X,C),
  asserta(X),
  on_backtracking(retract(X)),
  not(falsum).
```
• An example:
  
  \text{dom}(X) \rightarrow p(X); q(X).
  
  p(X) \rightarrow r(X); s(X).
  
  q(a) \rightarrow \text{falsum}.
  
  r(a) \rightarrow \text{falsum}.
  
  s(a) \rightarrow \text{falsum}.
  
  \text{dom}(a).

• A call to \text{satisfiable} then goes about searching for a model as follows ...
• There is a bug in this program, if the consequents of clauses contain unsatisfiable predicates.

• Example:

\[
\begin{align*}
\text{true}\quad & \rightarrow \quad p(X); \ q(X).
\text{p(a)}\quad & \rightarrow \quad \text{false}.
\text{p(b)}\quad & \rightarrow \quad \text{false}.
\end{align*}
\]
• Solution: limit clauses to those which are range-restricted: every variable in the consequent of a clause occurs in the antecedent as well.

• Fact

*Every set of clauses can be transformed into an equisatisfiable set of range-restricted clauses.*

• This general type of theorem is very important in logic.
The transformation process in this case

true---\rightarrow p(X); q(X).
p(a)---\rightarrow \text{false.}
q(b)---\rightarrow \text{false.}

\Rightarrow

\text{dom}(X) ---\rightarrow p(X); q(X).
p(a)---\rightarrow \text{false.}
q(b)---\rightarrow \text{false.}

\text{dom}(a).
\text{dom}(b).
• The transformation in general:
  • Any true → C which contains x₁, ..., xₙ, is replaced with dom(x₁) ∧ ... ∧ dom(xₙ) → C
  • Any other A → C with x₁, ..., xₙ occurring in C but not A is replaced with A ∧ dom(x₁) ∧ ... ∧ dom(xₙ) → C
  • For every constant c in any clause, the clause true → dom(c) is added. (Make sure there is at least one of these.)
  • For every n-ary function-symbol f in any clause, the clause dom(x₁) ∧ ... ∧ dom(xₙ) → dom(f(x₁, ..., xₙ)) is added.
• A problematic case:
  
  true---> p(a).
  p(X)---> p(f(X)).
  p(f(X)), p(g(X))---> falsum.
  p(X)---> p(g(X)).

• These clauses are unsatisfiable
• But satisfiable will generate an infinite sequence of atoms.
• What we should do is generate these things level-by-level.
  
  p(a)
  p(f(a))  p(g(a))
  p(f(f(a)))  p(f(g(a)))  p(g(f(a)))  p(g(g(a)))
  ...

• Here is the code that fixes the problem ...
satisfiable_level:- satisfiable_level(1).
satisfiable_level(L):-
    is_violated_level(L),
    !,
    on_backtracking(clean_level(L)),
    satisfy_level(L),
    L1 is L + 1,
    satisfiable_level(L1).
satisfiable_level(_L).
clean_level(L):-
    retract(generated(L, _X)),
    fail.
clean_level(_L).

satisfy_level(L):-
    generated(L, C),
    not(C), !,
    satisfy(C),
    satisfy_level(L),
    satisfy_level(L).

is_violated_level(L):-
    is_violated(C),
    not(generated(L, C)),
    fail.

is_violated_level(L):-
    generated(L, _X).
- We would like to prove facts about the blocks world using SATCHMO
- The plan is to translate the axioms we have directly into SATCHMO clauses
- There are just two problems...
• Problem 1: the treatment of equality.
• Suppose a clause contains the positive literal
  
  \( a = \text{table} \)
• SATCHMO will attempt to satisfy it by asserting it to the database.
• Such an attempt will result in the error-message:
  
  ! Attempt to redefine a system predicate
• We need to prevent such assertions.
• The solution is to replace the call to assert in satisfy by a call to my_assert, defined by:

\[
\text{my_asserta}(X=Y):-
\quad \text{not}(X=Y),!, \text{fail}.
\]

\[
\text{my_asserta}(X):- \text{asserta}(X).
\]

• This works by preventing the assertion of false identity statements.

• We even exploit the fact that Prolog makes the unique names assumption.
• Problem 2: returning the plan.
• Concretely, when we try to run the program, we can determine that, the above clauses are unsatisfiable, which shows us that

\[ \exists s \left( \text{situation}(s) \land \text{on}(a, b, s) \land \text{on}(b, c, s) \right) \]

is a theorem
• But we want a witness for \( s \)!
• This problem can be solved by modifying my_assert so that it saves important values.

• Just add, as the first clause in the definition

\[
\text{my_asserta}(\text{falsum}(X)):\neg
\]

\[
!,
\]

\[
\text{asserta}(\text{return_value}(X)),
\]

\[
\text{asserta}(\text{falsum}(X)).
\]

• and ...
... change satisfy to

\[
satisfy(C):-
\]
\[
  \text{component}(X,C),
\]
\[
  \text{my_asserta}(X),
\]
\[
  \text{on_backtracking}(\text{retract}(X)),
\]
\[
  \text{not}(\text{falsum}),
\]
\[
  \text{not}(\text{falsum}(_)).
\]

and change the critical clause in the problem to

\[
situation(S),\text{on}(a,b,S),\text{on}(b,c,S)---\rightarrow \text{falsum}(S).
\]
• The program can now be called as follows:

1 ?- [equality_satchmo].
% equality_satchmo compiled 0.02 sec, 4,540 bytes true.

2 ?- [eg_blocks].
% eg_blocks compiled 0.00 sec, 3,560 bytes true.

3 ?- satisfiable_level.
false.

4 ?- return_value(S).
S = do(move(a, table, b), do(move(b, table, c),
   do(move(a, b, table), s0))).