Today

School of informatics

- Propositional definite clauses ctd
- Monotone functions and power sets
- Completeness of the inference system.
- Forward chaining algorithm for derivability
- Relation to Prolog, Prolog search trees

Recall

We are interested in theories (as given by axioms), or equivalently logic programs, where each statement is a definite clause.
We only allow formulas of the shape p for some proposition p (ie an atomic statement), or

$$p_1 \wedge \cdots \wedge p_n \rightarrow q$$

where each p_i, q is an atomic statement.
We have the notion of when a basic statement q follows logically from set of such clauses $\mathcal{S}$,

$$\mathcal{S} \models q$$

defined by assignments of truth values to the atoms, and truth tables for the connectives.

Inference System

We also have a Inference System for these (propositional) definite clauses. Given a set $\mathcal{S}$ of definite clauses, we have:

- **Axiom** any statement in $\mathcal{S}$.
- **Inference rules** These allow us to infer the statement below the line from statements above the line, matching the patterns shown:
  - **MP**: $\dfrac{P \qquad P \rightarrow Q}{Q}$
  - **andI**: $\dfrac{P \qquad Q}{P \wedge Q}$

These rules can be seen to be sound, using truth tables — if the formulae above the line are true, then so is the formulae below.

Completeness of Inference System

It's surprising that this inference system is complete for the problem of showing whether a goal follows from $\mathcal{S}$.
The claim is that

$$\textit{if } \mathcal{S} \models q, \textit{ then } \mathcal{S} \vdash q$$

– if q follows logically, then there is a derivation using this inference system.

We'll now look at why this is the case.

Sets of sets

Given any set X, there is the corresponding set of (all) subsets of X, the power set of X, written $\mathcal{P}(X)$.
Example: $\mathcal{P}(\{\ 1, 2, 3\ \})$ is a set of eight sets:

$$\{\ \{\ \},\{\ 1\ \},\{\ 2\ \},\{\ 3\ \},\{\ 1, 2\ \},\{\ 1, 3\ \},\{\ 2, 3\ \},\{\ 1, 2, 3\ \}\ \}$$

We can consider the power set of an infinite set also.
Given any set of sets Y (not necessarily a power set), we define the intersection of Y, $\bigcap Y$, as the elements that belong to *every* set in Y.

$$x \in \bigcap Y \quad \text{if and only if} \quad \forall Z\ (Z \in Y \rightarrow x \in Z)$$

Example $\bigcap\{\ \{\ 1, 2, 3\ \},\{\ 1, 3\ \},\{\ 2, 3\ \}\ \} = \{\ 3\ \}$

Monotone functions

The standard notion of one set being a subset of another ($X \subseteq Y$) just means that everything that is in X is also in Y:

$$\forall x\ x \in X \rightarrow x \in Y.$$

Suppose that we have a function $f : X \rightarrow Y$, where X, Y are both sets of sets. (The notation $f : X \rightarrow Y$ says that f is a function that relates for any $x \in X$ a unique corresponding $f(x) \in Y$). Such a function f is said to be monotone just when the following holds for every pair of sets $X_1, X_2 \in X$:

$$X_1 \subseteq X_2 \rightarrow f(X_1) \subseteq f(X_2)$$

Examples

Consider the following $f_i : \mathcal{P}(\{\ 1, 2, 3\ \}) \rightarrow \mathcal{P}(\{\ 1, 2, 3\ \})$.

- $f_1(Y) = Y \cup \{\ 1\ \}$ is monotone
  ($A \cup B$ is the set that collects all the elements of A and B)
- $f_2(Y) = \begin{cases} \{\ 1\ \} & \text{if } 1 \in Y \\ \{\ \} & \text{otherwise} \end{cases}$ is monotone
- $f_3(Y) = \begin{cases} \{\ \} & \text{if } 1 \in Y \\ \{\ 1\ \} & \text{otherwise} \end{cases}$ is not monotone

Definite clauses and monotone functions

School of informatics

We can think of using our inference system to do forward inference.

- Any unit clauses (atoms $p_i \in \mathcal{S}$) are given as true, and are provable as axioms.
- We can deduce that a new atom r is true whenever we have deduced that $p_1, \ldots, p_n$ true, and $p_1 \wedge \cdots \wedge p_n \rightarrow r$ is in $\mathcal{S}$. A derivation here uses **andI** $n-1$ times and **MP** once.

So we can build larger and larger sets of atoms p_i that follow logically from the given $\mathcal{S}$, and also have derivations.
Suppose that A is the set of the atoms $p_1, p_2, \ldots$ appearing in $\mathcal{S}$.
Define $f : \mathcal{P}(A) \rightarrow \mathcal{P}(A)$ as follows.

Definite clauses and monotone functions

Let $Base = \{\ p_i : p_i \in \mathcal{S}\ \}$ be the set of all the atoms that appear in unit clauses in $\mathcal{S}$.
Then define

$$\begin{aligned} f(Y) \quad = \quad & Y \cup Base \\ & \cup \{\ a \in A : \quad (s_1 \wedge \cdots \wedge s_n \rightarrow a) \text{ is in } \mathcal{S}, \\ & \qquad\qquad s_1 \in Y, \ldots, s_n \in Y\ \} \end{aligned}$$

That is, $f(Y)$ adds to Y all the unit clause atoms, and also adds the right hand side (using the logic representation of clause) of any clauses in $\mathcal{S}$ where every atom in the left hand side is already in Y.

A check shows that $f : \mathcal{P}(A) \rightarrow \mathcal{P}(A)$ is monotone.

Using the function

School of informatics

We can now look for an interesting set of atoms.
Consider what happens when we look successively at

$$\{\ \}, f(\{\ \}), f(f(\{\ \})), f(f(f(\{\ \}))), \ldots$$

At each application of f, the resultant set either get bigger, or stays the same size (has the same number of elements).
Suppose there are exactly k atoms in the statements $\mathcal{S}$. That means that by the time we apply f $k+1$ times, there must be some place in the chain where the application of f returns the same set:

$$f(X) = X$$

Such an X is called a fixed point of the function f. If we apply f successively until we find an X with this property, we have found the least fixed point of f.

Example

Given:

$$\begin{aligned} \mathit{cold} &\rightarrow \mathit{wet} \\ \mathit{wet} \wedge \mathit{wet} &\rightarrow \mathit{scotland} \end{aligned}$$

we get that $f(\{\ \}) = \{\ \}$, and $\{\ \}$ is the least fixed point.

This tells us that no atoms follow logically from these statements.)

Example

School of informatics

In Prolog syntax:

```
cold.
wet :- cold.
dry :- dry.
scotland:- wet, cold.
```

$$
\begin{aligned}
f(\{\ \}) &= \{\ cold\ \} \\
f(\{\ cold\ \}) &= \{\ cold, wet\ \} \\
f(\{\ cold, wet\ \}) &= \{\ cold, wet, scotland\ \} \\
f(\{\ cold, wet, scotland\ \}) &= \{\ cold, wet, scotland\ \}
\end{aligned}
$$

So the least fixed point is $\{\ cold, wet, scotland\ \}$.

Claim

Claim:

> *the least fixed point is the set of all atoms that follow logically from the program/theory.*

Proof: We have already seen that the function f preserves truth, so each application of f gives a set of atoms that are true, if every statement in $\mathcal{S}$ is true.

In the other direction, suppose that X is the least fixed point. We want to show that if $\mathcal{S} \models q$, then $q \in X$.
We can show this by showing the following:

$$\text{if } \neg(q \in X), \text{ then } \neg(\mathcal{S} \models q).$$

Proof ctd

So, suppose $\neg(q \in X)$. To show $\neg(\mathcal{S} \models q)$, we need to find an interpretation $\mathcal{I}$ (an assignment of true, false to the atoms), such that $\mathcal{I}$ makes everything in $\mathcal{S}$ true, and q false.
Use this interpretation:

$$\mathcal{I}(p) = \begin{cases} T & \text{if } p \in X \\ F & \text{otherwise} \end{cases}$$

This makes q false. Now check that every statement in $\mathcal{S}$ is true, on this interpretation (ie, $\mathcal{I} \models \phi$ for every $\phi \in \mathcal{S}$). There are two cases, depending on the form of the definite clause in question.

Proof ctd

1. If p is a unit clause, then $p \in X$, so $\mathcal{I} \models p$.
2. If $\phi = p_1 \wedge \cdots \wedge p_n \rightarrow r$ is in $\mathcal{S}$, then:
   either some p_i is not in X, so $\mathcal{I} \models \phi$
   or all p_i are in X; and then $r \in f(X) = X$, so $\mathcal{I} \models r$, and thus $\mathcal{I} \models \phi$.

This concludes the proof.

Complete Inference System

Using this result, we get that

> *The inference system above is complete for showing $\mathcal{S} \models q$ for atomic queries q.*
> *This is because if $\mathcal{S} \models q$, then $q \in X$ where X is the least fixed point. But we have seen that each application of f gives a set of provable atoms; since the fixed point is reached after finitely many applications, every $a \in X$ has a derivation.*

This also suggests a forward chaining algorithm that will be *complete* (as an inference procedure) to compute whether $\mathcal{S} \vdash q$.

Computing the fixed point

There is an obvious iterative algorithm that lets us compute the sets $\{\ \}, f(\{\ \}), f(f(\{\ \})), \ldots$.
For a given set Y, $f(Y)$ is computed (after the first step, which is easy) by looping through the clauses to find the set of extra elements to add; add the extras after checking all the clauses.
Iterate this, until we find $f(X) = X$, and X is the fixed point.

To check if $\mathcal{S} \vdash q$, compute the fixed point X, and return the truth value of $q \in X$.

Completeness of the inference procedure

School of informatics

We want to show that if $\mathcal{S} \vdash q$, then the inference procedure will give us a derivation.
The procedure we have described does not explicitly construct derivations, however. So it is not complete in that sense. It *is* complete in the sense that if $\mathcal{S} \vdash q$, the procedure will return true.

Note that the procedure always terminates (why?). If $\mathcal{S} \vdash q$, then from soundness of the inference system $\mathcal{S} \models q$, and so $q \in X$. If we have correctly computed the fixed point, then we will have the right answer.
In fact, the procedure is a decision procedure that decides whether $\mathcal{S} \vdash q$; it always terminates with the correct answer.

Complexity of Inference

The algorithm as given above can be made more efficient. Suppose the number of atoms is a, and the number of clauses is c.
The loop is executed at most a times. Inside the loop, for each clause, we have at most a checks to do – make these constant time checks – plus constant overhead. Checking $X = f(X)$ can be bounded by a. So, each loop is bounded by ac. Overall bound is $a^2.c$.

It is known that there is a decision procedure for $\mathcal{S} \vdash q$ that runs in *linear* time with respect to the size of the program $\mathcal{S}$ (the number of occurrences of atoms in $\mathcal{S}$).
So this problem is much easier than the general problem of deducibility in propositional logic.

Backchaining algorithm

Recall that the Prolog algorithm uses backchaining, and is in fact incomplete. This is not so surprising, since it was designed for use in the more general situation, where we use predicate calculus, and there are good pragmatic reasons to sacrifice completeness.
The Prolog backchaining algorithm can be altered to keep track of possible looping, by keeping a record of atoms that have already been seen. This also gives a decision procedure for the problem; a linear-time version of backward chaining is also possible.

Example

Atoms derived at stage n are in red. Stage 0:

$$
\begin{aligned}
\mathit{chicago} &\rightarrow \mathit{windy} \\
\mathit{edinburgh} &\rightarrow \mathit{windy} \\
\mathit{edinburgh} &\rightarrow \mathit{scotland} \\
\mathit{scotland} &\rightarrow \mathit{rainy} \\
\mathit{windy} \wedge \mathit{rainy} &\rightarrow \mathit{insideOutUmbrella} \\
&\quad\ \mathit{edinburgh}
\end{aligned}
$$

Example (2)

Atoms derived at stage n are in red. Stage 1:

$$
\begin{aligned}
\mathit{chicago} &\rightarrow \mathit{windy} \\
\mathit{edinburgh} &\rightarrow \mathit{windy} \\
\mathit{edinburgh} &\rightarrow \mathit{scotland} \\
\mathit{scotland} &\rightarrow \mathit{rainy} \\
\mathit{windy} \wedge \mathit{rainy} &\rightarrow \mathit{insideOutUmbrella} \\
&\quad\ \mathit{edinburgh}
\end{aligned}
$$

Example (3)

Atoms derived at stage n are in red. Stage 2:

$$
\begin{aligned}
\mathit{chicago} &\rightarrow \mathit{windy} \\
\mathit{edinburgh} &\rightarrow \mathit{windy} \\
\mathit{edinburgh} &\rightarrow \mathit{scotland} \\
\mathit{scotland} &\rightarrow \mathit{rainy} \\
\mathit{windy} \wedge \mathit{rainy} &\rightarrow \mathit{insideOutUmbrella} \\
&\quad\ \mathit{edinburgh}
\end{aligned}
$$

Example (4)

Atoms derived at stage n are in red. Stage 3:

$$
\begin{aligned}
\mathit{chicago} &\longrightarrow \mathit{windy} \\
\mathit{edinburgh} &\longrightarrow \mathit{windy} \\
\mathit{edinburgh} &\longrightarrow \mathit{scotland} \\
\mathit{scotland} &\longrightarrow \mathit{rainy} \\
\mathit{windy} \wedge \mathit{rainy} &\longrightarrow \mathit{insideOutUmbrella} \\
&\qquad\quad \mathit{edinburgh}
\end{aligned}
$$

Example (5)

Atoms derived at stage n are in red. Stage 4:

$$
\begin{aligned}
\mathit{chicago} &\longrightarrow \mathit{windy} \\
\mathit{edinburgh} &\longrightarrow \mathit{windy} \\
\mathit{edinburgh} &\longrightarrow \mathit{scotland} \\
\mathit{scotland} &\longrightarrow \mathit{rainy} \\
\mathit{windy} \wedge \mathit{rainy} &\longrightarrow \mathit{insideOutUmbrella} \\
&\qquad\quad \mathit{edinburgh}
\end{aligned}
$$

Nothing new appears after this. So *chicago* is **not** derivable from the axioms; all the other atoms are.
This algorithm is not that used in Prolog – but we can use the least fixed point idea to analyse the meaning of predicate definite clause programs (pure Prolog).

Notation

School of informatics

- Goal $g_1, \ldots, g_m$ is a logical consequence of $F_1, \ldots, F_n$:

$$F_1, \ldots, F_n \models g_1 \wedge \cdots \wedge g_m$$

- Goal $g_1, \ldots, g_m$ is derivable from $F_1, \ldots, F_n$:

$$F_1, \ldots, F_n \vdash g_1 \wedge \cdots \wedge g_m$$

  i.e. there is some derivation of $g_1 \wedge \cdots \wedge g_m$ from axioms $F_1, \ldots, F_n$

- Goal $g_1, \ldots, g_m$ is Prolog derivable from $F_1, \ldots, F_n$:

$$F_1, \ldots, F_n \vdash_{Prolog} g_1 \wedge \cdots \wedge g_m$$

  i.e. Prolog search succeeds.

Fundamental Properties

School of informatics

Correctness of Prolog proof search

$F_1, \ldots, F_n \vdash_{Prolog} g_1 \wedge \cdots \wedge g_m$ implies $F_1, \ldots, F_n \vdash g_1 \wedge \cdots \wedge g_m$

Soundness of Inference System

$F_1, \ldots, F_n \vdash g_1 \wedge \cdots \wedge g_m$ implies $F_1, \ldots, F_n \models g_1 \wedge \cdots \wedge g_m$

Completeness of Inference System

$F_1, \ldots, F_n \models g_1 \wedge \cdots \wedge g_m$ implies $F_1, \ldots, F_n \vdash g_1 \wedge \cdots \wedge g_m$

Incompleteness of Prolog proof search

$F_1, \ldots, F_n \vdash g_1 \wedge \cdots \wedge g_m$ does not imply

$F_1, \ldots, F_n \vdash_{Prolog} g_1 \wedge \cdots \wedge g_m$

Fundamental Properties 2

School of informatics

Correctness of Prolog proof search: straightforward, next lecture

$F_1, \ldots, F_n \vdash_{Prolog} g_1 \wedge \cdots \wedge g_m$ implies $F_1, \ldots, F_n \vdash g_1 \wedge \cdots \wedge g_m$

Soundness of Inference System: not so hard, next lecture

$F_1, \ldots, F_n \vdash g_1 \wedge \cdots \wedge g_m$ implies $F_1, \ldots, F_n \models g_1 \wedge \cdots \wedge g_m$

Completeness of Inference System: propositional case argued today

$F_1, \ldots, F_n \models g_1 \wedge \cdots \wedge g_m$ implies $F_1, \ldots, F_n \vdash g_1 \wedge \cdots \wedge g_m$

Incompleteness of Prolog proof search:

take easy propositional example

$F_1, \ldots, F_n \vdash g_1 \wedge \cdots \wedge g_m$ does not imply

$F_1, \ldots, F_n \vdash_{Prolog} g_1 \wedge \cdots \wedge g_m$

Summary

School of informatics

- Propositional definite clauses ctd
- Monotone functions and power sets
- Completeness of the inference system.
- Forward chaining algorithm for derivability