Today

- Unification Algorithm and Occurs Check
- Inference System of definite clause logic
- General existence of least fixed points

Recall: Definite Clause Logic

Suppose we restrict the statements we consider as follows:

- Drop quantifiers (but keep variables)
- Drop $\neg$, $\vee$ (but keep $\wedge$, $\rightarrow$).
- Only allow formulas of the shape $p(t_1, \ldots, t_n)$ for some predicate p (ie an atomic statement), or

$$A_1 \wedge \cdots \wedge A_n \rightarrow B$$

  where each A_i is an atomic statement.

  This defines the Definite Clauses.

Inference System for Definite Clause Logic

The given definite clauses are taken as axioms. We use a single inference rule, Backchain:

$$\frac{p_1\theta, \;\; p_2\theta, \; \ldots, \; p_n\theta, \;\; (p_1 \wedge p_2 \wedge \ldots \wedge p_n \Rightarrow q)}{q'\theta} \text{where } \theta \text{ is mgu of } q', q$$

Given a query $(\exists X) r(X)$, see if $r(X)$ unifies with the "head" formula of a definite clause, with unifier θ. If so, top-down search will look for justifications of $p_1\theta, \;\; p_2\theta, \; \ldots, \; p_n\theta$.

Unification revisited

Recall that a most general unifier (mgu) of terms t_1, t_2 is a substitution S such that

- $t_1 S = t_2 S$ (it's a unifier),
- and for any other substitution S', if $t_1 S' = t_2 S'$, then $S' \preccurlyeq S$ (ie, for some other substitution T, $S' = S \circ T$;
  S is most general).

Consider how to design an algorithm to compute an mgu for t_1, t_2; we proceed by working through the term structure of the terms involved, and building up a unifier incrementally, using the composition of substitutions we saw earlier.

Unification ctd

Suppose the syntax of terms uses the following:

$$Term \ ::= \ \textbf{Cst}\ String \mid \textbf{Var}\ Int \mid \textbf{Fn}\ (String, Term\ List)$$

Some cases are easy,

- two constants unify with the identity subn if the are the same, and otherwise do not unify.
- two variables $\mathtt{v_m}, \mathtt{v_n}$ always unify with unifier $\{\ \mathtt{v_m}/\mathtt{v_n}\ \}$.
- a variable always unifies with a constant
- What about unifying a variable $\mathtt{v_n}$ with a term of the form $\mathtt{f}(\ldots)$?

Occurs check

It's tempting to think that the substitution $\{ \mathtt{v_n/t} \}$ will always unify $\mathtt{v_n}, \mathtt{t}$. But think of the case where $\mathtt{v_n}$ occurs in $\mathtt{t}$; is there a unifier $\mathtt{S}$ such that

$$\mathtt{v_n} = \mathtt{f(v_n)}?$$

If we try $\mathtt{S} = \{ \mathtt{v_n/f(v_n)} \}$ on both sides we get:

$$\mathtt{f(v_n)} = \mathtt{f(f(v_n))}$$

– we end up with different terms. So the simple solution is not right. In fact, with the standard understanding of the set of terms as given by the grammar definition, there is *no* substitution that makes these terms the same. In general, the unifier of $\mathtt{v_n}, \mathtt{t}$ is

- $\{ \mathtt{v_n/t} \}$ if v_n does not occur in t
- does not exist, if v_n occurs in t

What about unification of two terms both starting with function symbols, $\mathtt{f}_1(\mathtt{t}_1, \ldots, \mathtt{t}_\mathtt{n}), \mathtt{f}_2(\mathtt{u}_1, \ldots, \mathtt{u}_\mathtt{m})$?

- If $\mathtt{f}_1 \neq \mathtt{f}_2$, or $\mathtt{n} \neq \mathtt{m}$, then unification fails.
- Otherwise unify successively $\mathtt{t}_1, \mathtt{u}_1$ then $\mathtt{t}_2, \mathtt{u}_2 \ldots$, at each stage applying any substitution found to the remaining terms.

For example, how unify $\mathtt{f}(\mathtt{v}_1, \mathtt{f}(\mathtt{v}_1)) = \mathtt{f}(\mathtt{h}(\mathtt{v}_2), \mathtt{v}_3)$?
$\mathtt{f}$ is the same in both cases, so there are two problems to solve:

- $\mathtt{v}_1 = \mathtt{h}(\mathtt{v}_2)$ has unifier $\{\ \mathtt{v}_1/\mathtt{h}(\mathtt{v}_2)\ \}$; apply to second problem, to get
- $\mathtt{f}(\mathtt{h}(\mathtt{v}_2)) = \mathtt{v}_3$, with unifier $\{\ \mathtt{v}_3/\mathtt{f}(\mathtt{h}(\mathtt{v}_2))\ \}$, which composes with the first subn to give $\{\ \mathtt{v}_1/\mathtt{h}(\mathtt{v}_2), \mathtt{v}_3/\mathtt{f}(\mathtt{h}(\mathtt{v}_2))\ \}$.

Unification algorithm

It is tricky to get the unification algorithm right. However, it has been done. A correct implementation, given t_1, t_2, returns either failure, or a mgu.

Early algorithms were very inefficient – linear time algorithms are known for computing mgus. The main problem is the occurs check; terms involved can get very large when combining substitutions ...

In practice, most Prolog implementations do *not* include the occurs check in basic unification; but they usually have a version with the occurs check also.

```
| ?- X = f(X).
X = f(f(f(f(f(f(f(f(f(f(...)))))))))) ?
yes
| ?- unify_with_occurs_check(X,f(X)).
no
```

Unification algorithm:

Rule-based version of algorithm
(following exposition of Temur Kutsia).

General form of rules:

$$P; \rho \Longrightarrow Q; \theta \text{ or}$$
$$P; \rho \Longrightarrow \perp$$

where

- $\perp$ is failure (non-unification)
- ρ, θ are substitutions
- P, Q are lists of pairs of expressions:
  $\{ (E_1, F_1), \ldots, (E_n, F_n) \}$

Trivial:

$$\{\ (S, S)\ \} \cup P; \theta \Longrightarrow P; \theta$$

Decomposition:

$$\{\ (f(s_1, \ldots, s_n), f(t_1, \ldots, t_n))\ \} \cup P; \rho \ \Longrightarrow$$
$$\{\ (s_1, t_1), \ldots, (s_n, t_n)\ \} \cup P; \rho$$

Symbol clash

$$\{\ (f(s_1, \ldots, s_n), g(t_1, \ldots, t_n))\ \} \cup P; \rho \Longrightarrow \bot$$
$$\text{if } f \neq g$$

A similar case is needed if f can be used with different numbers of arguments.

Unification rules (2)

Orient

$$\{\ (t, x)\ \} \cup P; \rho \Longrightarrow \{\ (x, t)\ \} \cup P; \rho$$

if t is not a variable

Occurs check

$$\{\ (x, t)\ \} \cup P; \rho \Longrightarrow \perp$$

if t occurs in t, and $x \neq t$

Variable elimination

$$\{\ (x, t)\ \} \cup P'; \rho \Longrightarrow P'\theta; \rho\, o\, \theta$$

if x does not occur in t, and $\theta = \{\ x/t\ \}$

Unification algorithm

To unify expressions E_1, E_2:

- Start with $\{\ (E_1, E_2)\ \}; \{\ \}$
- Apply unification rules successively.

Unification algorithm: properties

- The algorithm always terminates, either with $\bot$, or $\{\,\}; \rho$.
- Soundness If the algorithm terminates with $\{\,\}; \rho$, then ρ is a unifier of the input expressions.
- Completeness If θ is a unifier for input expressions, then the algorithm finds a unifier ρ such that $\theta \preccurlyeq \rho$.
- MGU So: If input expressions are unifiable, then the algorithm returns a Most General Unifier (MGU).

Can we unify $\mathtt{p(X, f(X, Y), g(f(Y, X)))}$ and $\mathtt{p(c, Z, g(Z))}$?

$\{\ \mathtt{p(X, f(X, Y), g(f(Y, X)))},\ \mathtt{p(c, Z, g(Z))}\ \}; \{\,\}$	
$\{\ \mathtt{(X, c)},\ \mathtt{(f(X, Y), Z)},\ \mathtt{(g(f(Y, X)), g(Z))}\ \}; \{\,\}$	Decomp
$\{\ \mathtt{(f(X, Y), Z)}\{\ \mathtt{X/c}\ \},\ \mathtt{(g(f(Y, X)), g(Z))}\{\ \mathtt{X/c}\ \}\ \}; \{\ \mathtt{X/c}\ \}$	VarElim
$\{\ \mathtt{(f(c, Y), Z)}\ \mathtt{(g(f(Y, c)), g(Z))}\ \}; \{\ \mathtt{X/c}\ \}$	Apply subs
$\{\ \mathtt{(Z, f(c, Y))}\ \mathtt{(g(f(Y, c)), g(Z))}\ \}; \{\ \mathtt{X/c}\ \}$	align
$\{\ \mathtt{(g(f(Y, c)), g(Z))}\ \}\{\ \mathtt{Z/f(c, Y)}\ \}; \{\ \mathtt{X/c}\ \}\mathrm{o}\{\ \mathtt{Z/f(c, Y)}\ \}$	VarElim
$\{\ \mathtt{(g(f(Y, c)), g(f(c, Y)))}\ \}; \{\ \mathtt{X/c, Z/f(c, Y)}\ \}$	Apply subs
$\{\ \mathtt{(f(Y, c), f(c, Y))}\ \}; \{\ \mathtt{X/c, Z/f(c, Y)}\ \}$	Decomp
$\{\ \mathtt{(Y, c)},\ \mathtt{(c, Y)}\ \}; \{\ \mathtt{X/c, Z/f(c, Y)}\ \}$	Decomp
$\{\ \mathtt{(c, Y)}\{\ \mathtt{Y/c}\ \}\ \}; \{\ \mathtt{X/c, Z/f(c, Y)}\ \}\mathrm{o}\{\ \mathtt{Y/c}\ \}$	VarElim
$\{\ \mathtt{(c, c)}\ \}; \{\ \mathtt{X/c, Y/c, Z/f(c, Y)}\ \}$	Apply subs
$\{\,\}; \{\ \mathtt{X/c, Y/c, Z/f(c, Y)}\ \}$	Trivial

Fixed point revisited

To think of models of definite clause programs, we use a general property monotone functions defined over $\mathcal{P}(X)$. Recall that a fixed point of $f : \mathcal{P}(X) \rightarrow \mathcal{P}(X)$ is a set $Y \subseteq X$ such that $f(Y) = Y$.

A least fixed point of f is a fixed point that is smaller than any other fixed point of f, ie Y is a least fixed point (lfp) if

– it is a fixed point, and

– if Z is also a fixed point of f, then $Y \subseteq Z$.

There is a useful property of *monotone* functions defined as above:

> *Theorem: If $f : \mathcal{P}(X) \rightarrow \mathcal{P}(X)$ is monotone, then f has a least fixed point.*

We will use this to characterise the true statements that follow from a set of definite clauses.

LFP continued

In the case where X is finite, it's easy to see that successive computation of $f(\{\ \}), f(f(\{\ \})), f(f(f(\{\ \}))), \ldots$ will reach a fixed point.

Even in the finite case, there may be many fixed points. For example, given a simple program

```
a.
b :- c, a.
c :- d.
```

the corresponding $f : \mathcal{P}(\{\ a, b, c, d\ \}) \to \mathcal{P}(\{\ a, b, c, d\ \})$ has the lfp $\{\ a\ \}$.
Note that $\{\ a, b, c, d\ \}$ is also a fixed point.

In fact, any set of definite clauses is logically *consistent*, that is there is some model for the statements.

This is because we can interpret *every* atomic statement as being true. Then every clause will also be true, as you can check.

However, this is usually not the intended interpretation of definite clauses — think of definition of `parent/2` for example. Thus it is important to look at the least fixed point.

Proof of lfp property

We saw the intersection operation on sets before. It has the following properties:

1. If Y is a set of sets, then for all $Z \in Y$, $\bigcap Y \subseteq Z$.
   ($\bigcap Y$ is a lower bound for sets in Y)
2. If Y is a set of sets, and for every $Z \in Y, W \subseteq Z$
   (ie, W is a lower bound for sets in Y), then $W \subseteq \bigcap Y$.
   (Thus $\bigcap Y$ is the greatest lower bound for sets in Y)

We use these properties to find the lfp of a given monotone f.

Proof of lfp ctd

Let $Fix = \bigcap\{\ Y \in X \mid f(Y) \subseteq Y\ \}$.
First claim is that Fix is a fixed point of f; show this in two parts,
part 1: $f(Fix) \subseteq Fix$, then **part 2:** $Fix \subseteq f(Fix)$.
part 1.
Take some Z in the set $\{\ Y \in X \mid f(Y) \subseteq Y\ \}$.

We have $Fix \subseteq Z$	by property 1 of $\bigcap$
so $f(Fix) \subseteq f(Z)$	since f is monotone
Also $f(Z) \subseteq Z$	since $Z \in \{\ Y \in X \mid f(Y) \subseteq Y\ \}$
so $f(Fix) \subseteq Z$	by transitivity of $\subseteq$.

Since this holds for **any** Z in the set,
$f(Fix) \subseteq \bigcap\{\ Y \in X \mid f(Y) \subseteq Y\ \}$ by property 2 of $\bigcap$.

Thus $f(Fix) \subseteq Fix$.

Proof of lfp property ctd

Part 2

Now that we have $f(Fix) \subseteq Fix$, since f is monotone, we get $f(f(Fix)) \subseteq f(Fix)$;
this means that $f(Fix)$ is in the set $\{\ Y \in X \mid f(Y) \subseteq Y\ \}$, and so $Fix \subseteq f(Fix)$, by property 1 of $\bigcap$.
We now know that Fix is a fixed point of f.

Part 3

The final part of the claim is that Fix is the *least* fixed point of f.
This part is easy;
suppose Z is a fixed point: $Z = f(Z)$. Then $f(Z) \subseteq Z$, and $Z \in \{\ Y \in X \mid f(Y) \subseteq Y\ \}$, so that $Fix \subseteq Z$.

Summary

- Unification Algorithm and Occurs Check
- Inference System of definite clause logic
- General existence of least fixed points