Complete Knowledge Assumption

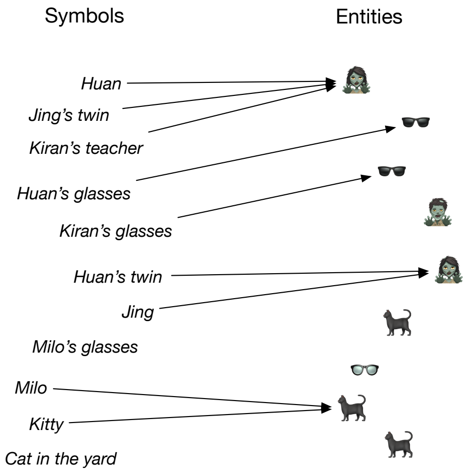
- Often you want to assume that your knowledge is complete.
- Example: assume that a database of what students are enrolled in a course is complete. We don't want to have to state all negative enrolment facts!
- The definite clause language is monotonic: adding clauses can't invalidate a previous conclusion.
- Under the complete knowledge assumption, the system is non-monotonic: adding clauses can invalidate a previous conclusion.

Equality

Equality is a special predicate symbol with a standard domain-independent intended interpretation.

- Suppose interpretation $I = \langle D, \phi, \pi \rangle$.
- t_1 and t_2 are ground terms then $t_1 = t_2$ is true in interpretation I if t_1 and t_2 denote the same individual. That is, $t_1 = t_2$ if $\phi(t_1)$ is the same as $\phi(t_2)$.
- $t_1 \neq t_2$ when t_1 and t_2 denote different individuals.
- Example:
  $D = \{$✂, ☎, ✎$\}$.
  $\phi(phone) =$ ☎, $\phi(pencil) =$ ✎, $\phi(telephone) =$ ☎
  What equalities and inequalities hold?
  $phone = telephone$, $phone = phone$, $pencil = pencil$,
  $telephone = telephone$
  $pencil \neq phone$, $pencil \neq telephone$
- Equality does not mean similarity!

Equality



Jing is Huan's twin. $(=)$
Jing is not Kiran's teacher. $(\neq)$

Properties of Equality

Equality is:

- Reflexive: $X = X$
- Symmetric: if $X = Y$ then $Y = X$
- Transitive: if $X = Y$ and $Y = Z$ then $X = Z$

For each n-ary function symbol f

$$f(X_1, \ldots, X_n) = f(Y_1, \ldots, Y_n) \text{ if } X_1 = Y_1 \text{ and } \cdots \text{ and } X_n = Y_n.$$

For each n-ary predicate symbol p

$$p(X_1, \ldots, X_n) \text{ if } p(Y_1, \ldots, Y_n) \text{ and } X_1 = Y_1 \text{ and } \cdots \text{ and } X_n = Y_n.$$

Unique Names Assumption

- Suppose the only clauses for *enrolled* are

  $enrolled(sam, cs222)$

  $enrolled(chris, cs222)$

  $enrolled(sam, cs873)$

  To conclude $\neg enrolled(chris, cs873)$, what do we need to assume?
  - All other enrolled facts are false
  - Inequalities:

$$sam \neq chris \wedge cs873 \neq cs222$$

- The unique names assumption (UNA) is the assumption that distinct ground terms denote different individuals.

Inequality as a subgoal

- What should the following query return?

$$?- \quad X \neq 4.$$

- What should the following query return?

$$?- \quad X \neq 4, \; X = 7.$$

- What should the following query return?

$$?- \quad X \neq 4, \; X = 4.$$

- Prolog has 3 different inequalities that differ on examples like these:

```
\==        \=        dif()
```

They differ in cases where there are free variables, and terms unify but are not identical.

Prolog's 3 implementations of not-equals

- Prolog has 3 different inequalities:

  `\==` `\=` `dif()`

  which give same answers for variable-free queries, or when both sides are identical

  `a \== 3,` `a \= 3,` `dif(a,3)`

  all succceed.

  `a \== a,` `a \= a,` `dif(a,a)`

  all fail.
- They give different answers when there is a free variable.
  `\==` means "not identical". `a \== X` succeeds
  `\=` means "not unifiable". `a \= X` fails
  `dif` is less procedural and more logical

Implementing dif

- *dif*(X, Y)
  - all instances fail when X and Y are identical
  - all instances succeed when X and Y do not unify
  - otherwise some instance succeed and some fail
- To implement *dif*(X, Y) in the body of a clause:
  - Select leftmost clause — unless it is a *dif* which cannot be determined to fail or succeed (delay dif calls)
  - Return the *dif* calls not resolved.
- Consider the calls:

```
dif(X,4), X=7.
dif(X,4), X=4.
dif(X,4), dif(X,7).
```

Example of dif

```
passed_two_courses(S) :-
    dif(C1,C2),
    passed(S, C1),
    passed(S, C2).
passed(S,C) :-
    grade(S,C,M),
    M >= 50.
grade(sam,engl101,87).
grade(sam,phys191,89).
```

Other predicates, such as #<, work similarly;

```
use_module(library(clpfd)).
    % https://www.swi-prolog.org/man/clpfd.html
X #< Y, Y #< Z, Z #< X.
```

Constraint programming systems provide sophisticated constraint solving.

Completion of a knowledge base: propositional case

- Suppose the rules for atom a are

$$a \leftarrow b_1.$$
$$\vdots$$
$$a \leftarrow b_n.$$

  equivalently $a \leftarrow b_1 \vee \ldots \vee b_n$.

- Under the Complete Knowledge Assumption, if a is true, one of the b_i must be true:

$$a \rightarrow b_1 \vee \ldots \vee b_n.$$

- Thus, the clauses for a mean

$$a \leftrightarrow b_1 \vee \ldots \vee b_n$$

Clark Normal Form

The Clark normal form of the clause

$$p(t_1, \ldots, t_k) \leftarrow B.$$

is the clause

$$p(V_1, \ldots, V_k) \leftarrow \exists W_1 \ldots \exists W_m \; V_1 = t_1 \land \ldots \land V_k = t_k \land B.$$

where

- $V_1, \ldots, V_k$ are k variables that did not appear in the original clause
- $W_1, \ldots, W_m$ are the original variables in the clause.
- When the clause is an atomic clause, B is *true*.
- Often can be simplified by replacing $\exists W \; V = W \land p(W)$ with $P(V)$.

Clark normal form

For the clauses

$$student(mary).$$
$$student(sam).$$
$$student(X) \leftarrow undergrad(X).$$

the Clark normal form is

$$student(V) \leftarrow V = mary.$$
$$student(V) \leftarrow V = sam.$$
$$student(V) \leftarrow \exists X \; V = X \wedge undergrad(X).$$

Clark's Completion

Suppose all of the clauses for p are put into Clark normal form, with the same set of introduced variables, giving

$$p(V_1, \ldots, V_k) \leftarrow B_1.$$
$$\vdots$$
$$p(V_1, \ldots, V_k) \leftarrow B_n.$$

which is equivalent to

$$p(V_1, \ldots, V_k) \leftarrow B_1 \vee \ldots \vee B_n.$$

Clark's completion of predicate p is the equivalence

$$\forall V_1 \ldots \forall V_k \; p(V_1, \ldots, V_k) \leftrightarrow B_1 \vee \ldots \vee B_n$$

If there are no clauses for p, the completion results in

$$\forall V_1 \ldots \forall V_k \; p(V_1, \ldots, V_k) \leftrightarrow \mathit{false}$$

Clark's completion of a knowledge base consists of the completion of every predicate symbol along the unique names assumption.

Completion Example

Consider the recursive definition:

$$passed\_each([\,], St, MinPass).$$

$$passed\_each([C|R], St, MinPass) \leftarrow$$
$$passed(St, C, MinPass) \wedge$$
$$passed\_each(R, St, MinPass).$$

In Clark normal form, this can be written as

$$passed\_each(L, S, M) \leftarrow L = [\,].$$

$$passed\_each(L, S, M) \leftarrow$$
$$\exists C\ \exists R\ \ L = [C|R] \wedge passed(S, C, M) \wedge passed\_each(R, S, M).$$

Here we renamed the variables as appropriate. Thus, Clark's completion of *passed_each* is

$$\forall L\ \forall S\ \forall M\ \ passed\_each(L, S, M) \leftrightarrow L = [\,] \vee$$
$$\exists C\ \exists R\ \ L = [C|R] \wedge passed(S, C, M) \wedge passed\_each(R, S, M).$$

Clark's Completion of a KB

- Clark's completion of a knowledge base consists of the completion of every predicate.
- The completion of an n-ary predicate p with no clauses is $p(V_1, \ldots, V_n) \leftrightarrow \mathit{false}$.
- You can interpret negations in the body of clauses. $\sim a$ means a is false under the complete knowledge assumption. $\sim a$ is replaced by $\neg a$ in the completion. This is negation as failure.

Completion example

$$p \leftarrow q \wedge \sim r.$$
$$p \leftarrow s.$$
$$q \leftarrow \sim s.$$
$$r \leftarrow \sim t.$$
$$t.$$
$$s \leftarrow w.$$

Completion:

$$p \leftrightarrow q \wedge \neg r \vee s.$$
$$q \leftrightarrow \neg s.$$
$$r \leftrightarrow \neg t.$$
$$t \leftrightarrow true.$$
$$s \leftrightarrow w.$$
$$w \leftrightarrow false.$$

Defining $empty\_course$

Given database of:

- $course(C)$ that is true if C is a course
- $enrolled(S, C)$ that is true if student S is enrolled in course C.

Define $empty\_course(C)$ that is true if there are no students enrolled in course C.

- Using negation as failure, $empty\_course(C)$ can be defined by

$$empty\_course(C) \leftarrow course(C) \wedge \sim has\_enrollment(C).$$

$$has\_enrollment(C) \leftarrow enrolled(S, C).$$

- The completion of this is:

$$\forall C \; empty\_course(C) \iff course(C) \wedge \neg has\_enrollment(C).$$

$$\forall C \; has\_enrollment(C) \iff \exists S \; enrolled(S, C).$$

Bottom-up negation as failure interpreter

$C := \{\}$;
repeat
    either
        select $r \in KB$ such that
            r is "$h \leftarrow b_1 \wedge \ldots \wedge b_m$"
            $b_i \in C$ for all i, and
            $h \notin C$;
        $C := C \cup \{h\}$
    or
        select h such that for every rule "$h \leftarrow b_1 \wedge \ldots \wedge b_m$" $\in KB$
                either for some $b_i, \sim b_i \in C$
                or some $b_i = \sim g$ and $g \in C$
        $C := C \cup \{\sim h\}$
until no more selections are possible

Negation as failure example

$$p \leftarrow q \wedge \sim r.$$

$$p \leftarrow s.$$

$$q \leftarrow \sim s.$$

$$r \leftarrow \sim t.$$

$$t.$$

$$s \leftarrow w.$$

Top-Down negation as failure proof procedure

- If the proof for a fails, you can conclude $\sim a$.
- Failure can be defined recursively:
  Suppose you have rules for atom a:

$$
\begin{aligned}
a &\leftarrow b_1 \\
&\vdots \\
a &\leftarrow b_n
\end{aligned}
$$

  If each body b_i fails, a fails.
- A body fails if one of the conjuncts in the body fails.
- Note that you need *finite* failure. Example $p \leftarrow p$.

Floundering

$$p(X) \leftarrow \sim q(X) \wedge r(X).$$
$$q(a).$$
$$q(b).$$
$$r(d).$$
$$\text{ask } \; p(X).$$

- What is the answer to the query?
- How can a top-down proof procedure find the answer?
- Delay the subgoal until it is bound enough.
  Sometimes it never gets bound enough — "floundering".

Problematic Cases

$$p(X) \leftarrow \sim q(X)$$
$$q(X) \leftarrow \sim r(X)$$
$$r(a)$$
ask $p(X)$.

- What is the answer?
- What does delaying do?
- How can this be implemented?