Complete Knowledge Assumption (CKA)

Sometimes you want to assume that a database of facts is complete. Any fact not listed is false.

Example: Assume that a database of *enrolled* relations is complete. Then you can define *empty_course*.

Example: Assume a database of video segments is complete.

The definite clause RRS is monotonic: adding clauses doesn't invalidate a previous conclusion.

With the complete knowledge assumption, the system is nonmonotonic: a conclusion can be invalidated by adding more clauses.

CKA: propositional case

Suppose the rules for atom a are

$$a \leftarrow b_1.$$

$$\ldots$$

$$a \leftarrow b_n.$$

or equivalently: $a \leftarrow b_1 \vee \ldots \vee b_n$

Under the CKA, if a is true, one of the b_i must be true:

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

Under the CKA, the clauses for a mean Clark's completion:

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

CKA: Ground Database

Example: Consider the relation defined by:

$$student(mary).$$

$$student(john).$$

$$student(ying).$$

The CKA specifies these three are the only students:

$$student(X) \leftrightarrow X = mary \vee X = john \vee X = ying.$$

To conclude $\neg student(alan)$, you have to be able to prove

$$alan \neq mary \wedge alan \neq john \wedge alan \neq ying$$

This needs the unique names assumption.

Clark Normal Form

The Clark normal form of the clause:

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

is the clause

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

where $V_1, \ldots, V_k$ are k different variables that did not appear in the original clause.

$W_1, \ldots, W_m$ are the original variables in the clause.

Clark normal form: example

- The Clark normal form of:

$$room(C, room208) \leftarrow$$
$$cs\_course(C) \wedge enrollment(C, E) \wedge E < 120.$$

is

$$room(X, Y) \leftarrow \exists C \exists E\ X = C \wedge Y = room208 \wedge$$
$$cs\_course(C) \wedge enrollment(C, E) \wedge E < 120.$$

Clark's Completion of a Predicate

Put all of the clauses for p into Clark normal form, with the same set of introduced variables:

$$p(V_1, \ldots, V_k) \leftarrow B_1$$

$$\vdots$$

$$p(V_1, \ldots, V_k) \leftarrow B_n$$

This is the same as: $p(V_1, \ldots, V_k) \leftarrow B_1 \vee \ldots \vee B_n$.

Clark's completion of p is the equivalence

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

That is, $p(V_1, \ldots, V_k)$ is true if and only if one B_i is true.

Clark's Completion Example

Given the *mem* function:

$$mem(X, [X|T]).$$

$$mem(X, [H|T]) \leftarrow mem(X, T).$$

the completion is

$$mem(X, Y) \iff (\exists T\ Y = [X|T]) \vee$$

$$(\exists H \exists T\ Y = [H|T] \wedge mem(X, T))$$

Clark's Completion of a KB

- Clark's completion of a knowledge base consists of the completion of every predicate symbol, along with the axioms for equality and inequality.
- If you have a predicate p defined by no clauses in the knowledge base, the completion is $p \leftrightarrow \mathit{false}$. That is, $\neg p$.
- You can interpret negations in the bodies of clauses. $\sim p$ means that p is false under the Complete Knowledge Assumption. This is called negation as failure.

Using negation as failure

Previously we couldn't define $empty\_course(C)$ from a database of $enrolled(S, C)$.

This can be defined using negation as failure:

$$
\begin{aligned}
& empty\_course(C) \leftarrow \\
& \quad course(C) \wedge \\
& \quad \sim has\_Enrollment(C). \\
& has\_Enrollment(C) \leftarrow \\
& \quad enrolled(S, C).
\end{aligned}
$$

Bottom-up NAF proof procedure

$C := \{\}$;

repeat

either select "$h \leftarrow b_1 \wedge \ldots \wedge b_m$"$\in KB$ such that

$b_i \in C$ 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 NAF Procedure

If the proof for a fails, you can conclude $\sim a$.

Failure can be defined recursively.

Suppose you have rules for atom a:

$$a \leftarrow b_1$$

$$\vdots$$

$$a \leftarrow b_n$$

If each body b_i fails, a fails.

A body fails if one of the conjuncts in the body fails.

Note that you require *finite* failure. Example: $p \leftarrow p$.

Free Variables in Negation as Failure

Example:

$$p(X) \leftarrow \sim q(X) \wedge r(X).$$

$$q(a).$$

$$q(b).$$

$$r(d).$$

There is only one answer to the query $?p(X)$, namely $X = d$.

For calls to negation as failure with free variables, you need to delay negation as failure goals that contain free variables until the variables become bound.

Floundering Goals

If the variables never become bound, a negated goal **flounders.**

In this case you can't conclude anything about the goal.

Example: Consider the clauses:

$$p(X) \leftarrow \sim q(X)$$

$$q(X) \leftarrow \sim r(X)$$

$$r(a)$$

and the query

$$?p(X).$$