Why not assume KB includes all positive facts, and everything else is false (similar to unique name assumption)?

Examples:
- If I haven't stated that two rooms are adjacent, assume they are not.
- If I haven't stated that Jim is Mary's father, assume he is not.

We don't want to state negative facts in KB, but we want to ask queries of form $\neg p$ and use $\neg p$ in the body of a rule.

How can we formalize complete knowledge assumption?

Horne Clauses
- Allowed negative information to be expressed.
- Some things are true.
- Some things are false, using the negative information.
- Some things it just doesn't know.
- System can conclude now that.

Complete Knowledge Assumption

Overview
- Approach 1: Clark's Completion
- Approach 2: Negation as Failure
- Complete Knowledge Assumption
Variable Case: Example

- Example
  - student(mary)
  - student(john)
  - student(ying)

- Same as
  - student(X) ← X = mary
  - student(X) ← X = john
  - student(X) ← X = ying

- Note that `=' sign could mean equality

- Collect them all together and you get
  - student(X) ← X = mary ∨ X = john ∨ X = ying

- Completion is
  - student(X) ↔ X = mary ∨ X = john ∨ X = ying

Clark's Completion

- Typically only used with Datalog
- If you have predicate defined by no clauses in KB
  - Clark's Completion: a ↔ (b₁ ∨ ... ∨ bₙ)
  - Clark's Normal Form
  - You have equivalently a → (b₁ ∨ ... ∨ bₙ)

- If you have
  - Typically only used with Datalog

Approach 2: Negation as Failure

Approach 1: Clark's Completion

Complete Knowledge Assumption
Using Clark's Completion

• Typically just used with Datalog

Clark Completion though requires:

- Disjunctive and negative knowledge
- Usually also assume UNA or need axioms for equality

Can be applied to just some of the predicates, not necessarily all

- Only use rules from KB that have predicate on left hand side
- Do not use ones in which it is on the right hand side

Several Clauses

• Say if you have in Clark Normal form

\[
 p(V_1, \ldots, V_n) \leftarrow C_1 \\
 \vdots \\
 p(V_1, \ldots, V_n) \leftarrow C_n
\]

• Clark completion of \( p \) is

\[
 p(V_1, \ldots, V_n) \leftrightarrow C_1 \lor \ldots \lor C_n
\]

- Note that each \( C_i \) might have a number of conjunctions to it for its
  variables

- Before putting it together, make sure each part doesn't have any other
  variables in common other than the \( V_i \)’s

Example:

\[
 b \lor \forall x \in A \lor \ldots \lor \exists y \in A \leftrightarrow \left( \forall x \rightarrow (\forall y \rightarrow A)d \right)\]

\[
 b \lor \forall x \in A \lor \ldots \lor \exists y \in A \leftrightarrow \left( \forall y \rightarrow A)d
\]

\[
 b \rightarrow \left( \forall x \rightarrow A)d
\]
Overview

• Complete Knowledge Assumption
  • Approach 1: Clark's Completion
  ⇒ Approach 2: Negation as Failure

Can I prove \( \neg ls(s(0),0) \)?

\[ \text{yes} \quad \lor \quad \neg ls(A,B) \quad \text{yes} \quad \lor \quad 0=s(X) \quad \lor \quad 0=s(Y) \]

Unique Name Assume

Example

• Can be used with recursive predicates

\[ \text{lt}(0,s(X)) \]
\[ \text{lt}(s(X),s(Y)) \quad \leftarrow \quad \text{ls}(X,Y) \]

Clark Normal Form:

\[ \text{lt}(A,B) \quad \leftarrow \quad A=0 \quad \land \quad B=s(X) \]

\[ \text{lt}(A,B) \quad \leftarrow \quad A=s(X) \quad \land \quad B=s(Y) \quad \land \quad \text{ls}(X,Y) \]

Clark Completion:

\[ \text{ls}(A,B) \quad \leftrightarrow \quad (A=0 \quad \land \quad B=s(X)) \quad \lor \quad (A=s(X) \quad \land \quad B=s(Y) \quad \land \quad \text{ls}(X,Y)) \]

New rule to add to KB:

\[ (A=0 \quad \land \quad B=s(X)) \quad \lor \quad (A=s(X) \quad \land \quad B=s(Y) \quad \land \quad \text{ls}(X,Y)) \quad \leftarrow \quad \text{ls}(A,B) \]

Convert to CNF:

\[ (A=0 \quad \land \quad B=s(X)) \quad \lor \quad (A=s(X) \quad \land \quad B=s(Y) \quad \land \quad \text{ls}(X,Y)) \quad \lor \quad \neg \text{ls}(A,B) \]

New Axioms

\[ A=0 \quad \lor \quad A=s(X) \quad \lor \quad \neg \text{ls}(A,B) \]
\[ 0=s(X) \quad \lor \quad 0=s(Y) \quad \lor \quad \neg \text{ls}(A,B) \]
\[ B=s(X) \quad \lor \quad B=s(Y) \quad \lor \quad \neg \text{ls}(A,B) \]

Continue
Variables and Delaying

• Must be careful about variables
  - Similar to inequality, need to delay till variables are instantiated
  - Prolog doesn't do this

Example:

\[ KB = \{ p(X) \leftarrow \neg q(X) \land r(X), q(a), q(b), r(b), r(d) \} \]

Modifying a Top-Down Reasoning Procedure

• When see \( \neg p \), do recursive proof
  - If you can prove \( p \), then fail; otherwise succeed

Example:

\[ KB = \{ p(X) \leftarrow r(X) \land \neg q(X), q(a), q(b), r(b), r(d) \} \]

Negation as Failure

• A simpler way to make the CKA is to use negation as failure

- \( d \sim \neg \) allows negative queries
- \( d \sim \neg \) allows \( \neg \) to be used in bodies of clausal clauses
- \( d \sim \neg \) allows \( \neg \) to fail in rules
- \( d \sim \neg \) makes the CKA into a negative reasoner
- \( d \sim \neg \) is a simpler way to make the CKA to use negation as failure