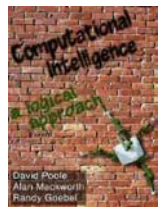
Chapter 7: Beyond Definite Knowledge

- Equality
- Complete Knowledge Assumption
- Integrity Constraints



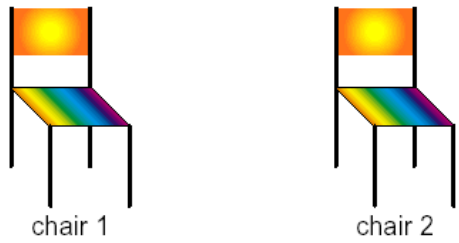
D. Poole, A. Mackworth, and R. Goebel, *Computational Intelligence: A Logical Approach*, Oxford University Press, January 1998

Equality

- Sometimes two terms denote the same individual.
- Example: Clark Kent & Superman. 4 * 4 & 11+5. The projector we used last week & this projector.
- Ground term t_1 equals ground term t_2, written $t_1 = t_2$, is true in interpretation I if t_1 and t_2 denote the same individual in interpretation I.

Equality doesn't mean similarity

WORLD of 2 Chairs



chair1 ≠ chair2

chair_on_right = chair2

chair_on_right is not similar to chair2, it is chair2.

Allowing Equality Assertions

- Without equality assertions, the only thing that is equal to a ground term is itself.

  This can be captured as though you had the assertion X = X. Explicit equality never needs to be used.

- If you allow equality assertions, you need to derive what follows from them. Either:
  - axiomatize equality like any other predicate
  - build special-purpose inference machinery for equality

Axiomatizing Equality

$X = X.$

$X = Y \leftarrow Y = X.$

$X = Z \leftarrow X = Y \wedge Y = Z.$

For each n-ary function symbol f there is a rule of the form

$f(X_1, \ldots, X_n) = f(Y_1, \ldots, Y_n) \leftarrow$

$X_1 = Y_1 \wedge \ldots \wedge X_n = Y_n.$

For each n-ary predicate symbol p, there is a rule of the form

$p(X_1, \ldots, X_n) \leftarrow$

$p(Y_1, \ldots, Y_n) \wedge X_1 = Y_1 \wedge \ldots \wedge X_n = Y_n.$

Special-Purpose Equality Reasoning

- paramodulation: if you have $t_1 = t_2$, then you can replace any occurrence of t_1 by t_2.
- Treat equality as a rewrite rule , substituting equals for equals.
- You select a canonical representation for each individual and rewrite all other representations into that representation.
- Example: treat the sequence of digits as the canonical representation of the number such as: 123267.
- Example: use the student number as the canonical representation for students.

Instead of axiomatizing the equality, we better use the UNA

Unique Names Assumption (UNA)

- The convention that different ground terms denote different individuals is the unique names assumption (false in RRS). for every pair of distinct ground terms t_1 and t_2, assume $t_1 \neq t_2$, where "≠" means "not equal to."
- Example: For each pair of courses, you don't want to have to state, math302 ≠ psyc303, …
- Example: Sometimes the unique names assumption is inappropriate, for example $3 + 7 \neq 2 * 5$ is wrong.

Axiomatizing Inequality for the UNA

- $c \neq c'$ for any distinct constants c and c'.
- $f(X_1, \ldots, X_n) \neq g(Y_1, \ldots, Y_m)$ for any distinct function symbols f and g.
- $f(X_1, \ldots, X_n) \neq f(Y_1, \ldots, Y_n) \leftarrow X_i \neq Y_i$, for any function symbol f . There are n instances of this schema for every n-ary function symbol f (one for each i such that $1 \leq i \leq n$).
- $f(X_1, \ldots, X_n) \neq c$ for any function symbol f and constant c.
- $t \neq X$ for any term t in which X appears (where t is not the term X).

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.
- The definite clause RRS is monotonic: adding clauses doesn't invalidate a previous conclusion.
- With the complete knowledge assumption and the UNA, the system is nonmonotonic: a conclusion can be invalidated by adding more clauses.

CKA: propositional case

Suppose the rules for atom a are

$a \leftarrow b_1$.

.

.

$a \leftarrow b_n$.

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

Under the CKA, if a is true in some interpretation, one of the b_i must be true in that interpretation (if all b_i's false then a cannot be infered which means a=false :

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

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

$a \leftrightarrow b_1 \vee \ldots \vee b_n$ *(if and only if)*

CKA: Ground Database

- Example: Consider the relation defined by:
  student(mary).
  student(john).
  student(ying).
  ?student(alan) "cannot conclude in RRS"

- The CKA specifies these three are the only students:
  $student(X) \leftrightarrow X = mary \vee X = john \vee X = ying.$

- To conclude : ¬student(alan), you have to be able to prove
  $alan \neq mary \wedge alan \neq john \wedge alan \neq ying$
  if this clause is added and true $\Rightarrow$*nonmonotonic*

- 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's Completion of a Predicate

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

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

  ...

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

- 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, $\mathbf{p(V_1, \ldots, V_k)}$ is true if and only if one B_i is true.

Example

- For the clauses:

  student(mary).
  student(john).
  student(ying).

The Clark normal form is:
student(V)← V = mary.
student(V) ← V = john.
student(V) ← V = ying. which is the same as:
Student(V) ← V=mary ∨ V=john ∨ V=ying.
The completion of the student predicate is:

student(V) ↔ V=mary ∨ V=john ∨ V=ying.

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 $\leftrightarrow$ false. That is, $\neg$ p.
- You can interpret negations in the bodies of clauses. ~p means that p is false under the Complete Knowledge
- Assumption. This is called negation as failure.

Negation as failure example

$p \leftarrow q \wedge \sim r.$
$p \leftarrow s.$
$q \leftarrow \sim s.$
$r \leftarrow \sim t.$
$t.$
$s \leftarrow w.$ *The completion of this KB is:*
$p \leftrightarrow (q \wedge \neg r) \vee \mathbf{s.}$
$\mathbf{q} \leftrightarrow \neg s.$
$\mathbf{r} \leftrightarrow \neg t$
$t \leftrightarrow$ true.
$s \leftrightarrow w.$
$w \leftrightarrow$ false (w not in the head).

Using negation as failure

- Previously in initial RRS we couldn't define empty_course(C) from a database of enrolled(S, C).
- This can be defined using negation as failure:

empty_course(C) ←
    course(C) ∧
    ~has_Enrollment(C).
has_Enrollment(C) ←
    enrolled(S, C).

Other Negation Failure

- empty_course(C) ← course(C) ^ ~enrolled(S,C).

Example:
course(cs422).
course(cs486).
enrolled(mary, cs422).
enrolled(sally, cs486).

the clause:
empty_course(cs422)← course(cs422) ∧
~enrolled(sally, cs422)
is an instance of this clause which leads to a contradiction!
(body = true, head = false)

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 $g \in C$
    $C := C \cup \{h\}$
until no more selections are possible

- Consider the clauses

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

- The following is a sequence of atoms added to C:

  t %(fact)
  $\sim r$
  $\sim w$ %(no clauses for w)
  $\sim s$
  q
  p.

Top-Down NAF Procedure

If the proof for a fails, you can conclude ~ a.

Failure can be defined recursively.

Suppose you have rules for atom a:

$a \leftarrow b_1$

…

$a \leftarrow b_n$

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: $p \leftarrow p$.

- Consider the clauses

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

  $p \leftarrow s.$

  $q \leftarrow \sim s.$

  $r \leftarrow \sim t.$

  $t.$

  $s \leftarrow w.$

Query: ?p

- your first try to prove q and to fail to prove r,
- To prove q you try to fail to prove s
- To prove s you try to prove w (w fails), etc. until you conclude p.

Integrity Constraints

- In the electrical domain, what if we predict that a light should be on, but observe that it isn't?
  What can we conclude?
- We will expand the definite clause language to include integrity constraints which are rules that imply *false,* where *false* is an atom that is false in all interpretations.
- This will allow us to make conclusions from a contradiction.
- A definite clause knowledge base is always consistent.
  This won't be true with the rules that imply *false*.

Horn clauses

- An integrity constraint is a clause of the form

  $$false \leftarrow a_1 \wedge \ldots \wedge a_k$$

- where the a_i are atoms and false is a special atom that is false in all interpretations.
- A Horn clause is either a definite clause or an integrity constraint.

Integrity Constraints in Databases

- Database designers can use integrity constraints to specify constraints that should never be violated.
- Example: A student can't have two different grades for the same course.

$$false \leftarrow Grade(St, Course, Gr_1) \wedge Grade(St, Course, Gr_2) \wedge Gr_1 \neq Gr_2.$$

- When false is derived, it can be used to debug the KB.