DB Updates & NonMonotonic Reasoning

CS240B Notes

Notes based on Section 10.1 of Advanced Database Systems—Morgan Kaufmann, 1997

C. Zaniolo, April 2002
Beyond Stratified Negation

- We need classes programs more powerful than those were negation and aggregates are stratified.
- The problem (at least in terms of fixpoint theory) is due to the non-monotonic nature of the implicit negation used in DBs and AI.
- Implicit negation: negative facts are inferred from the absence of the opposite conclusion, under the closed-world assumption.
- Nonmonotonic reasoning, and knowledge representation: a well-established research topic in AI. The concept of circumscription was followed by default theories and auto-epistemic logic; the concept of stable models is recent.
Open World and Closed World

- **Open World**: what is not part of the database or the program is assumed to be unknown.

- **Closed World**: what is not part of the database or the program is assumed to be false.

Databases and other information systems adopt the Closed World Assumption (CWA). If \( p \) is a base predicate with \( n \) arguments, then \( \neg p(a_1, \ldots, a_n) \) iff \( p(a_1, \ldots, a_n) \) is not true, i.e., it is not in the fact base.

*Unique name* axiom: no two constants in the database stand for the same semantic object.
Open vs. Closed World: example

The absence of coolguy(“Clark Kent”) database implies that \( \neg \text{coolguy(“Clark Kent”)}, \) even though the database contains a fact coolguy(“Super Man”).

For positive programs, the CWA is as follows: Let \( P \) be a positive program, then each atom \( a \in B_P \):

1. \( a \) is true iff \( a \in T_P^\omega(\emptyset) \)
2. \( \neg a \) is true iff \( a \notin T_P^\omega(\emptyset) \).

However the CWA for general programs (i.e., programs with negated goals) might lead to inconsistencies.
In the village, the barber shaves everyone who does not shave himself: Every villager, who does not shave himself, is shaved by the barber

\[
\text{shaves(barber, } X) \leftarrow \text{villager}(X), \neg\text{shaves}(X, X).
\]

\[
\text{shaves(miller, miller).}
\]

\[
\text{villager(miller).}
\]

\[
\text{villager(smith).}
\]

\[
\text{villager(barber).}
\]

There is no problem with villager(miller), who shaves himself, and therefore does not satisfies the body of the first rule.
Paradoxes and Contradictions: cont

1. For villager(smith), given that \( \text{shaves(smith, smith)} \) is not in our program, we can assume that
   \( \neg \text{shaves(smith, smith)} \); then, \( \text{shaves(barber, smith)} \) is derived that is consistent with with the negative assumptions made.

2. For villager(barber): under the assumption
   \( \neg \text{shaves(barber, barber)} \), the rule yields
   \( \text{shaves(barber, barber)} \) which contradicts the initial assumption.

3. If we do not initially assume \( \neg \text{shaves(barber, barber)} \), then we cannot derive \( \text{shaves(barber, barber)} \) using this program and by the CWA, we will have to assume
   \( \neg \text{shaves(barber, barber)} \), and end-up with a contradiction.
Stable Models

Programs that have Stable Models avoid self-contradictions\

**Stability Transformation.** Let $P$ a program and $I \subseteq B_P$ be an interpretation of $P$. Then $ground_M(P)$ denote the program obtained from $ground(P)$ by the following transformation:

1. remove every rule having as a goals some literal $\neg q$ with $q \in I$
2. remove all negated goals from the remaining rules.
3. Example, where $P = ground(P)$:

   \[
   p \leftarrow \neg q.
   \]
   \[
   q \leftarrow \neg p.
   \]
1. Let $P$ be a program with model $M$. $M$ is said to be a stable model for $P$, when $M$ is the least model of $\text{ground}_M(P)$.

2. $\text{ground}_M(P)$ is a positive program, by construction: so, its least model is $T^\omega(\emptyset)$, where $T$ denotes the immediate consequence operator for $\text{ground}_M(P)$.

3. Every stable model for $P$ is a minimal model for $P$ and a minimal fixpoint for $T_P$. 
Stable Models: properties

But minimal models or minimal fixpoints might not be stable models. Example: \( M = \{ a \} \) is the only model and fixpoint for the program:

\[
\begin{align*}
  r_1 &: a \leftarrow \neg a. \\
  r_2 &: a \leftarrow a.
\end{align*}
\]

1. This program has no stable model
2. A program can have zero stable models, one stable model or several stable models
3. Theorem: Given a negative Datalog program \( P \), deciding whether this has a stable model is \( \mathcal{NP} \)-complete
4. The existence of a stable model can depend on the database. For instance, the barber program has a unique stable model after we eliminate villager(\text{barber}).
Multiple Models

A program can have several stable models.

\[ p \leftarrow \neg q \]

\[ q \leftarrow \neg p \]

1. This has two stable models: \( M_1 = \{p\} \) and \( M_2 = \{q\} \).

2. With multiple models, one needs to decide what the intended semantics is: fin all models, or find one? We take the second interpretation, which leads to the concept of NonDeterminism.

3. Stratified Programs, however, always have a unique stable model.