Beyond Stratification

CS240B Notes

Notes based on Section 10.4 of Advanced Database Systems—Morgan Kaufmann, 1997

C. Zaniolo, April 2002
Many proposals—e.g., locally stratified programs and well founded models—on how to go beyond stratified negation.

Objective is to approach the power of stable model, without its exponential complexity.

Also the semantic well-formedness of the program can be determined from the rules (independent of the DB) as for stratified programs.

XY-stratification: is a particular class of locally stratified programs for which we also have a simple compile-time check, and an efficient implementation.

In fact, XY-stratified programs are particular Datalog₁ˢ programs.
Ancestors of marc and their generation gap expressed using the differential fixpoint:

\[
\begin{align*}
r_1 &: \text{delta}_\text{anc}(0, \text{marc}). \\
r_2 &: \text{delta}_\text{anc}(J + 1, Y) \leftarrow \text{delta}_\text{anc}(J, X), \text{parent}(Y, X), \\
     & \quad \neg \text{all}_\text{anc}(J, Y). \\
r_3 &: \text{all}_\text{anc}(J + 1, X) \leftarrow \text{all}_\text{anc}(J, X). \\
r_4 &: \text{all}_\text{anc}(J, X) \leftarrow \text{delta}_\text{anc}(J, X).
\end{align*}
\]
1. This program is locally stratified by the first argument in \texttt{anc} that serves as temporal argument.

2. The zero stratum consists of atoms of nonrecursive predicates such as \texttt{parent} and of atoms that unify with \texttt{all\_anc(0, X)} or \texttt{delta\_anc(0, X)}, where \texttt{X} can be any constant in the universe.

3. The $k^{th}$ stratum contains atoms \texttt{all\_anc(k, X)} and \texttt{delta\_anc(k, X)}.

Thus, this program is locally stratified, since the heads of recursive rules belong to strata that are one above those of their goals.
X-rules and Y-rules

- $r$ is an X-rule when the temporal argument (TA) in every recursive predicate in $r$ is the same variable (e.g., $J$),
- $r$ is a Y-rule when for some variable $J$
  1. the head of $r$ has $J + 1$ as its TA
  2. some goal of $r$ has as TA $J$, and
  3. the remaining recursive goals have either $J$ or $J + 1$ as their TAs.

XY-programs: Let $P$ be a set of rules defining mutually recursive predicates. Then we say that $P$ is an XY-program when:

1. Every recursive predicate of $P$ has a distinguished temporal argument.
2. Every recursive rule $r$ is either an X-rule or a Y-rule.
The Old and the New

Given an XY-program $P$, its bi-state program, $P_{bis}$, is computed as follows: For each $r \in P$,

1. Rename all the recursive predicates in $r$ that have the same temporal argument as the head of $r$ with the distinguished prefix `new_`

2. Rename all other occurrences of recursive predicates in $r$ with the distinguished prefix `old_`

3. Drop the temporal arguments from the recursive predicates.

\[
\begin{align*}
\text{new}_\text{delta}_\text{anc}(\text{marc}). \\
\text{new}_\text{delta}_\text{anc}(Y) & \leftarrow \text{old}_\text{delta}_\text{anc}(X), \text{parent}(Y, X), \neg\text{old}_\text{all}_\text{anc}(Y). \\
\text{new}_\text{all}_\text{anc}(X) & \leftarrow \text{new}_\text{delta}_\text{anc}(X). \\
\text{new}_\text{all}_\text{anc}(X) & \leftarrow \text{old}_\text{all}_\text{anc}(X).
\end{align*}
\]
**Definition:** Let $P$ be an XY-program. $P$ is said to be XY-stratified when $P_{bis}$ is a stratified program.

Our previous program is stratified with the following strata:

- $S_0 = \{\text{parent, old\_all\_anc, old\_delta\_anc}\}$,
- $S_1 = \{\text{new\_delta\_anc}\}$
- $S_2 = \{\text{new\_all\_anc}\}$

**Theorem:** Let $P$ be an XY-stratified program. Then $P$ has a unique stable model.
Computing the stable model of an XY-stratified program $P$

**Initialize:** Set $T = 0$ and insert the fact $\text{counter}(T)$.

Forever repeat the following two steps:

1. Apply the iterated fixpoint computation to the synchronized program $P_{\text{bis}}$, and for each recursive predicate $q$, compute $\text{new}_q$. Return the $\text{new}_q$ atoms so computed, after adding a temporal argument $T$ to these atoms; the value of $T$ is taken from $\text{counter}(T)$.

2. For each recursive predicate $q$, replace $\text{old}_q$ with $\text{new}_q$, computed in the previous step. Then, replace $\text{counter}(T)$ with $\text{counter}(T + 1)$.

- Copy rules
- When does the computation stop?
Classical Algorithms can be Expressed as XY-stratified programs

A simple example: Coalescing after Temporal Projection.

\[
\text{emp\_dep\_sal}(1001, \text{shoe}, 35000, 19920101, 19940101).
\]

\[
\text{emp\_dep\_sal}(1001, \text{shoe}, 36500, 19940101, 19960101).
\]

These two tuples must be merged.
Merging overlapping periods after a temporal projection

\[
e_{\text{hist}}(0, \text{Eno}, \text{Frm}, \text{To}) \leftarrow \text{emp}_{\text{dep}_{\text{sal}}}(0, \text{Eno}, D, S, \text{Frm}, \text{To}).
\]
\[
\text{overlap}(J + 1, \text{Eno}, \text{Frm}_1, \text{To}_1, \text{Frm}_2, \text{To}_2) \leftarrow \\
\quad \quad e_{\text{hist}}(J, \text{Eno}, \text{Frm}_1, \text{To}_1), \\
\quad \quad e_{\text{hist}}(J, \text{Eno}, \text{Frm}_2, \text{To}_2), \\
\quad \quad \text{Frm}_1 \leq \text{Frm}_2, \text{Frm}_2 \leq \text{To}_1, \\
\quad \quad \text{distinct}(\text{Frm}_1, \text{To}_1, \text{Frm}_2, \text{To}_2).
\]
\[
e_{\text{hist}}(J, \text{Eno}, \text{Frm}_1, \text{To}) \leftarrow \text{overlap}(J, \text{Eno}, \text{Frm}_1, \text{To}_1, \text{Frm}_2, \text{To}_2), \\
\quad \quad \text{select}_{\text{larger}}(\text{To}_1, \text{To}_2, \text{To}).
\]
\[
e_{\text{hist}}(J + 1, \text{Eno}, \text{Frm}, \text{To}) \leftarrow e_{\text{hist}}(J, \text{Eno}, \text{Frm}, \text{To}), \\
\quad \quad \text{overlap}(J + 1, _, _, _, _, _), \\
\quad \quad \neg \text{overlap}(J + 1, \text{Eno}, \text{Frm}, \text{To}, _, _), \\
\quad \quad \neg \text{overlap}(J + 1, \text{Eno}, _, _, \text{Frm}, \text{To}).
\]
\[
\text{final}_{\text{eHist}}(\text{Eno}, \text{Frm}, \text{To}) \leftarrow e_{\text{hist}}(J, \text{Eno}, \text{Frm}, \text{To}), \\
\quad \quad \neg e_{\text{hist}}(J + 1, _, _, _).
\]
Temporal Projection–auxiliary predicates

distinct(Frm1, To1, Frm2, To2) ← To1 ≠ To2.
distinct(Frm1, To1, Frm2, To2) ← Frm1 ≠ Frm2.
select_larger(X, Y, X) ← X ≥ Y.
select_larger(X, Y, Y) ← Y > X.