NonDeterministic Reasoning

CS240B Notes

Notes based on Section 10.6 of *Advanced Database Systems*—Morgan Kaufmann, 1997

C. Zaniolo, April 2002

Nondeterminism and FD constraints

With relation $\texttt{student}(\texttt{Name}, \texttt{Majr}, \texttt{Year})$, our university database contains the relation $\texttt{professor}(\texttt{Name}, \texttt{Majr})$. A toy database with only the following facts:

$\texttt{student}('\texttt{Jim Black}', \texttt{ee}, \texttt{senior}).$

$\texttt{professor}(\texttt{ohm}, \texttt{ee}).$

$\texttt{professor}(\texttt{bell}, \texttt{ee}).$

$$\texttt{elig\_adv}(\texttt{S}, \texttt{P}) \leftarrow \texttt{student}(\texttt{S}, \texttt{Majr}, \texttt{Year}),\ \texttt{professor}(\texttt{P}, \texttt{Majr}$$

We obtain:

$\texttt{elig\_adv}('\texttt{Jim Black}', \texttt{ohm}).$

$\texttt{elig\_adv}('\texttt{Jim Black}', \texttt{bell}).$

But, a student can only have one advisor!?

Choice Goals

Then, in a language such as LDL++ the goal $\texttt{choice}((\texttt{S}),(\texttt{P}))$ can be added to force the selection of a unique advisor, out of the eligible advisors, for a student:

Computation/selection of unique advisors by choice rules

$$\texttt{actual\_adv}(\texttt{S},\texttt{P}) \leftarrow \texttt{student}(\texttt{S},\texttt{Majr},\texttt{Levl}),\ \texttt{professor}(\texttt{P},\texttt{Majr}),\ \texttt{choice}((\texttt{S}),(\texttt{P})).$$

More declaratively, the goal $\texttt{choice}((\texttt{S}),(\texttt{P}))$ can also be viewed as enforcing a functional dependency (FD) $\texttt{S} \rightarrow \texttt{P}$; thus, in $\texttt{actual\_adv}$, the second column (professor name) is functionally dependent on the first one (student name).

Many Applications of Choice

Given the two relations $\texttt{boy}(\texttt{Bname})$, $\texttt{girl}(\texttt{Gname})$,:

Are there more boys than girls in our database?

$$
\begin{aligned}
\texttt{match}(\texttt{Bname}, \texttt{Gname}) \leftarrow \quad & \texttt{boy}(\texttt{Bname}),\ \texttt{girl}(\texttt{Gname}). \\
& \texttt{choice}((\texttt{Bname}), (\texttt{Gname})), \\
& \texttt{choice}((\texttt{Gname}), (\texttt{Bname})). \\
\texttt{matched\_boy}(\texttt{Bname}) \leftarrow \quad & \texttt{match}(\texttt{Bname}, \texttt{Gname}). \\
\texttt{moreboys} \leftarrow \quad & \texttt{boy}(\texttt{Bname}),\ \neg\texttt{matched\_boy}(\texttt{Bname}).
\end{aligned}
$$

Choice by Negation

$$\texttt{actual\_adv}(\texttt{S},\texttt{P}) \leftarrow \texttt{student}(\texttt{S},\texttt{Majr},\texttt{Yr}),\ \texttt{professor}(\texttt{P},\texttt{Majr}$$
$$\texttt{choice}((\texttt{S}),(\texttt{P})).$$

The stable version for the adivisor rule:

$$\texttt{actual\_adv}(\texttt{S},\texttt{P}) \leftarrow \texttt{student}(\texttt{S},\texttt{Majr},\texttt{Yr}),\ \texttt{professor}(\texttt{P},\texttt{Majr}$$
$$\texttt{chosen}(\texttt{S},\texttt{P}).$$
$$\texttt{chosen}(\texttt{S},\texttt{P}) \leftarrow \texttt{student}(\texttt{S},\texttt{Majr},\texttt{Yr}),\ \texttt{professor}(\texttt{P},\texttt{Majr}$$
$$\neg\texttt{diffChoice}(\texttt{S},\texttt{P}).$$
$$\texttt{diffChoice}(\texttt{S},\texttt{P}) \leftarrow \texttt{chosen}(\texttt{S},\texttt{P}'),\texttt{P} \neq \texttt{P}'.$$

This program has two stable models. One in which `ohm` is chosen as advisor of `JimBlack`, and the other where `bell` is chosen instead.

Choice Models

1. A program where the rules contain choice goals is called a *choice program*.
2. The semantics of a choice program P can be defined by transforming P into a program with negation, $SV(P)$, called the *stable version* of a choice program P.
3. $SV(P)$ exhibits a multiplicity of stable models, each obeying the FDs defined by the choice goals.
4. Each stable model for $SV(P)$ corresponds to an alternative set of answers for P and is called a *choice model* for P.

Properties

In general, the program $SV(P)$ generated by the transformation discussed above has the following properties:

- $SV(P)$ has one or more total stable models.
- The `chosen` atoms in each stable model of $SV(P)$ obey the FDs defined by the choice goals.

The stable models of $SV(P)$ are called *choice models* for P. Stratified Datalog programs with choice are in DB-PTIME: actually they can be implemented efficiently by producing `chosen` atoms one at a time and memorizing them in a table. The `diffchoice` atoms need not be computed and stored; rather, the goal $\neg$`diffchoice` can simply be checked dynamically against the table `chosen`.

Choice in Recursion

For instance, the following program computes the spanning tree, starting from the source node `a`, for a graph where an arc from node `b` to `d` is represented by the database fact $\mathtt{g}(\mathtt{b},\mathtt{d})$.

Computing a spanning tree

$$\mathtt{st}(\mathtt{root},\mathtt{a}).$$

$$\mathtt{st}(\mathtt{X},\mathtt{Y}) \leftarrow \quad \mathtt{st}(\_,\mathtt{X}),\ \mathtt{g}(\mathtt{X},\mathtt{Y}),\ \mathtt{Y} \neq \mathtt{a},\ \mathtt{choice}((\mathtt{Y}),(\mathtt{X})).$$

The goal $\mathtt{Y} \neq \mathtt{a}$ ensures that, in `st`, the end-node for the arc produced by the exit rule has an in-degree of one;
likewise, the goal $\mathtt{choice}((\mathtt{Y}),(\mathtt{X}))$ ensures that the end-nodes for the arcs generated by the recursive rule have an in-degree of one.

Ordering a Domain

The following program defines a total order for the elements of a set $\mathtt{d}(\mathtt{X})$ by constructing an immediate-successor relation for its elements ($\mathtt{root}$ is a distinguished new symbol):

$$\mathtt{ordered\_d}(\mathtt{root}, \mathtt{root}).$$

$$\mathtt{ordered\_d}(\mathtt{X}, \mathtt{Y}) \leftarrow \quad \mathtt{ordered\_d}(\_, \mathtt{X}),\ \mathtt{d}(\mathtt{Y}),\ \mathtt{choice}((\mathtt{X}), (\mathtt{Y})),\ \mathtt{choice}((\mathtt{Y}), (\mathtt{X})).$$

Once an arc $(\mathtt{X}, \mathtt{Y})$ is generated, this is the only arc leaving the source node $\mathtt{X}$ and the only arc entering the sink node $\mathtt{Y}$.

DB-PTIME without Total Order

1. stratified Datalog programs with choice are DB-PTIME complete, without having to assume that the universe is totally ordered (i.e., respecting the genericity assumption).
2. Here we accept any order. Since we accept any choice model, we have don‘t care non-determinism and the computation remains polynomial.
3. For certain queries, we might still a deterministic result: e.g., in the computation of aggregates which are commutative and associative.

Beyond Don't Care Non-Determinism

In many situations, we seek to satisfy a condition that holds or does not hold depending on the choice made. Thus, we might want to seek among the choice models one that satisfy the condition. Alternatively, we might make a choice and then backtrack to the next choice once we find that the condition does not hold. Thus an exponential computation is often required.
Hamiltonian path in a graph: A graph has a Hamiltonian path iff there is a simple path that visits all nodes exactly once.

$$\texttt{simplepath}(\texttt{root},\texttt{root}).$$

$$\texttt{simplepath}(\texttt{X},\texttt{Y}) \leftarrow \quad \texttt{simple\_path}(\_,\texttt{X}),\ \texttt{g}(\texttt{X},\texttt{Y}),\ \texttt{choice}((\texttt{X}),(\texttt{Y})),\ \texttt{choice}((\texttt{Y}),(\texttt{X})).$$

$$\texttt{nonhppath} \leftarrow \quad \texttt{n}(\texttt{X}),\ \neg\,\texttt{simplepath}(\_,\texttt{X}).$$

$$\texttt{q} \leftarrow \quad \neg\texttt{q},\ \texttt{nonhppath}.$$

Hamiltonian Path

$$\texttt{simplepath}(\texttt{root},\texttt{root}).$$

$$\texttt{simplepath}(\texttt{X},\texttt{Y}) \leftarrow \quad \texttt{simple\_path}(\_,\texttt{X}),\ \texttt{g}(\texttt{X},\texttt{Y}),\ \texttt{choice}((\texttt{X}),(\texttt{Y})),\ \texttt{choice}((\texttt{Y}),(\texttt{X})).$$

$$\texttt{nonhppath} \leftarrow \quad \texttt{n}(\texttt{X}),\ \neg\,\texttt{simplepath}(\_,\texttt{X}).$$

$$\texttt{q} \leftarrow \quad \neg\texttt{q},\ \texttt{nonhppath}.$$

If nonhppath is true in M, then rule $\texttt{q} \leftarrow \neg\texttt{q}$ must also be satisfied by M. Thus, M cannot be a stable model. Thus, this program has a stable model iff there exists a Hamiltonian path. Thus, deciding whether a stable model exists for a program is $\mathcal{NP}$-hard.