Homework 12

Zachary Moring

December 12, 2024

Code for the Lean portion is here: `https://github.com/zpm-bu/cs511-formal-methods/blob/assignments/lean/Homework/hw12.lean`

Exercise 1, Part (a) Argue as to whether or not second-order logic or higher is reuqired for any SQL operations you are familiar with. Is first-order logic sufficient for all SQL operations?

First-order logic is sufficient for the core SQL behavior, or anything that is directly based on Codd's relational calculus. The basic operations for `SELECT`, `FROM`, `JOIN`, and so on do not require second-order logic.

Second-order logic *may* be required for some implementation-specific features. An example is the `CREATE TABLE IF EXISTS ...` construct from MySQL and Postgres. This construct first performs a check to see if there exists any table with the given name, and then if not, performs the operation; in order to express this behavior in logical statements, we need to be able to quantify over the known tables. This requires at least second-order logic.

Since these are not universal, it's down to the individual student answer to make a decent argument. Their answer will depend on the specific examples they use in their argument.

Exercise 1, Part (b) Draw a diagram for the schema in the diagram language we used in lecture on December 3rd. Use filled circles for table vertices and empty circles for type vertices.

$$\begin{tikzcd}
\texttt{<STRING>} & \texttt{<INTEGER>} & \texttt{<DATE\_TIME>} \\
\circ & \circ & \circ \\
\bullet \arrow[u, "name"] & \bullet \arrow[l, "account"] \arrow[u, "amount"] \arrow[ur, "date"] \arrow[r, "cd\_code"] & \bullet \\
\texttt{ACCOUNTS} & \texttt{TRANSACTIONS} & \texttt{CREDIT\_DEBIT}
\end{tikzcd}$$

Recall that according to our convention, the “key” fields on a table don’t need arrows. If the students *do* draw the keys, they should just point to the correct type node; but they are not required.

Exercise 2, Part (a) List the 15 morphisms on $\mathcal{K}$ along with their domain and codomain.

$$\begin{aligned}
id_A &: A \to A \\
id_B &: B \to B \\
id_C &: C \to C \\
id_D &: D \to D \\
f &: A \to B \\
g &: A \to B \\
h &: A \to C \\
i &: A \to D \\
j &: A \to D \\
k &: B \to C \\
\ell &: D \to C \\
f \fatsemi k &: A \to C \\
g \fatsemi k &: A \to C \\
i \fatsemi \ell &: A \to C \\
j \fatsemi \ell &: A \to C
\end{aligned}$$

Exercise 2, Part (b) List the morphisms of $\mathcal{K}'$.

The morphisms here are quite simpler:

$$\begin{aligned}
id_A &: A \to A \\
&\vdots \\
f &: A \to B \\
k &: B \to C \\
i &: A \to D \\
\ell &: D \to C \\
f \mathbin{⨟} k &: A \to C
\end{aligned}$$

Graphically, $\mathcal{K}'$ is a commutative square:

$$\begin{tikzcd}
A \arrow[r, "f"] \arrow[d, "i"] & B \arrow[d, "k"] \\
D \arrow[r, "\ell"] & C
\end{tikzcd}$$

The key learning from this exercise is that under commutativity, diagrams are significantly simpler. In particular, h should entirely disappear.

Exercise 2, Part (c) Define a functor $F : \mathcal{V} \to \mathcal{K}$.

Probably the most obvious such functor:

$$\begin{tikzcd}
& B \arrow[dr, "k" description, ""{name=K}] & U \arrow[l, dashed, blue] \arrow[dr, "p" description, ""{name=P}] & \\
A \arrow[ur, bend left, "f" description] \arrow[ur, "g" description] \arrow[rr, "h" description] \arrow[dr, "i" description] \arrow[dr, bend right, "j" description] & & C & V \arrow[l, dashed, blue] \\
& D \arrow[ur, "\ell" description, ""{name=L}] & W \arrow[l, dashed, blue] \arrow[ur, "q" description, ""{name=Q}] &
\arrow[from=P, to=K, dashed, blue]
\arrow[from=Q, to=L, dashed, blue]
\end{tikzcd}$$

Or enumerated explicitly:

$$\begin{gathered}
F(U) \mapsto B \\
F(V) \mapsto C \\
F(W) \mapsto D \\
F(p) \mapsto k \\
F(q) \mapsto \ell
\end{gathered}$$

Exercise 2, Part (d) Argue why there cannot be such a functor.

First off, as Aaron pointed out, there actually can be. If you map all the points to the same target point and all the morphisms to the identity, you can construct a trivial functor for any structure that you like.

The nontrivial functor cannot be constructed because there is no 4-element cycle in $\mathcal{K}$, and thus no function from the cycle to $\mathcal{K}$ can preserve composition.