Problem 1: Denotational Semantics of a Relational Database Language. 50 pts

Consider an SQL like relational database programming language. Programs in this language manipulate a relational database. A relational database (RDB) stores a table (collection of tuples) for each relation name. Thus, each relation can be regarded as a collection of tuples.

(12 pts) Define a semantic algebra for this database domain which supports the following operations Note: all definitions should be in proper lambda calculus format and should take care of error situations. Alternatively, you could implement them in Prolog, in a manner similar to HW 6.

- **create**: creates an empty database.
- **retrieve(database, relname)**: given a relation name in a database it returns the table corresponding to that relation.
- **store(database, relname, table)**: given a relation name in a database and a table, it stores the table in the database under that name.

Assume that `table` has as its domain

\[
table = key \rightarrow onetuple
\]

and

\[
onetuple = key \times list
\]

where `key = \text{Nat}` (set of natural numbers) and `list = \text{Nat}^*` (list of natural numbers). Thus, a collection of tuples is modeled as a function. Each tuple is modeled as a pair, whose first element is the primary key and the second element is a list of remaining elements of the tuple.

(13 pts) Extend the semantic algebra with the following operations:

- **create-rel(database, relname)**: given a database and a relation name, it initializes this relation to an empty collection of tuples in the database.
- **access(database, relname, key)**: given a database, a relation name, and a key, it returns the tuple in the relation that matches the key.
- **update(database, relname, newtuple)**: given a database, a relation name, and a tuple, it adds or updates the relation relname with tuple `newtuple` where `newtuple` is of type `onetuple` (note that the key can be extracted from the `newtuple`; if a tuple with that key is already present, it is replaced, if not, the tuple `newtuple` is added).
Now consider part of the Grammar for expressions in the database language:

\[ C \in \text{Commands} \]
\[ E \in \text{Relational-Expression} \]
\[ C ::= R[D] := E \]
\[ E ::= E_1 \bowtie E_2 \mid E_1 - E_2 \mid \pi_n(E) \mid R(D) \]

with the meanings:

1. In the productions above, \( R[D] \) refers to the table corresponding to relation \( R \) in database \( D \). A relational expression \( E \) returns a table (collection of tuples).

2. \( R(D) := E \) computes result of relational expression \( E \) and stores under the relation name \( R \) in Database \( D \).

3. \( E_1 \bowtie E_2 \) is the usual relational database join of two relational expressions \( E_1 \) and \( E_2 \). Two tuples \((k, l_1)\) and \((k, l_2)\) that have the same key and that have been joined will appear as \((k, append(l_1, l_2))\) in the resulting relation.

4. \( E_1 - E_2 \) produces a relation that is obtained by removing from relational expression \( E_1 \) the tuples of relational expression \( E_2 \) that are also present in \( E_1 \).

5. \( \pi_n(E) \) takes a relational expression \( E \) as input and produces the relation consisting of 2 element tuples where the first element is the key in the tuples of relation returned by \( E \) and the second element is the \( n \)th element of the tuples of relation returned by \( E \).

(25 pts) Give the valuation function of \( C \) and \( E \). Note that some functions may be recursive.