CS393P: Program Synthesis Project Proposal

1. **Idea**

As proven in the Flash-fill paper, synthesis is a very powerful tool in the context of database management and manipulation. My idea for a project is similar to it in that it attempts to rely on little initial user input and find patterns in strings.

My high-level goal for this project is to create a synthesizer that can derive patterns from both an existing excel database and of future entries that may be of a different format, order, etc. Ideally, the synthesizer would be able to work through these differences and format the new string to the database.

Let’s look at an example to help visualize the procedure. Below is a sample database:

FIRST NAME LAST NAME PHONE NUMBER AGE

Steven Callahan 123-456-7890 20  
Jim White 444-444-4444 56

Now, let’s say a user had the following string which they wanted to automatically add to this database:

“JEAN, MARK 27”

The synthesizer would use a mix of substring permutations and type checks to find patterns in the entry, figuring out the input string follows the general format of “<string>, <string> <int>”. The synthesizer would also be able to figure out the format of each column: from left to right, the table follows the format <string> <string> <string> <int>.

First, because AGE and 27 are the only int variables, they can be trivially matched.

Second, though cross-checking the examples it can be seen “JEAN” and “MARL” match FIRST NAME and LAST NAME far better than PHONE NUMBER, so PHONE NUMBER would be left blank.

Finally, through placement of the comma the synthesizer would know JEAN is a last name and MARK is a first name, the signatures of which would be understood by the synthesizer and added to the correct columns. Some additional formatting would recognize the singular capital letter followed by many lowercase letters and format the input strings accordingly.

Thus, the table would add the following, now correctly formatted, entry:

FIRST NAME LAST NAME PHONE NUMBER AGE

Mark Jean (blank) 27

1. **DSL**

Given this synthesizer would be operating on databases, it would include specifications for different types of strings, ints, etc. that represent common database entries, especially those that could be easily mixed up. Such entries could include:

<string:first\_name>  
<string:last\_name>  
<string:phone\_number>  
<int:age>  
<string:email>  
etc.

Each one of these specifications would have its own associated regex that could be used to check if a candidate substring belongs in one of these categories.

These entries could be interpreted in an existing database through column names or column entries.

For more specific entries, a simple type specification will be given, such as <int> or <bool>. If the synthesizer is unsure of the typing of a particular column, it could attempt multiple different types

1. **Algorithmic Approach**

The algorithm would likely take a “skeleton and hole” approach seen in previous papers, whereby the input string would be concatenated as parts of it were assigned.

Let’s take the above example; the initial call to the synthesizer would be something of the following:

Synthesize(“JEAN, MARK 27”, [<string:first\_name>, <string:last\_name>, <string:phone\_number>, <int:age>])

After trivially finding the match between “27” and AGE, a recursive call could be made as such:

Synthesize(“JEAN, MARK”, [<string:first\_name>, <string:last\_name>, <string:phone\_number>])

With no clear match the synthesizer would instead compare existing entries in each column for which are “most like” the candidate entries, and in the process eliminate phone number as a candidate. A third recursive Synthesize() call will be made:

Synthesize(“JEAN, MARK”, [<string:first\_name>, <string:last\_name>])

Finally, given pattern recognition would see and interpret “<string>, <string>” as “<string:last\_name>, <string:first\_name>”, and finally return the matched values.

Now, when considering the proposed matches, a cost function will be used whereby strong matches with all columns filled will be preferred (i.e., keeping a column blank or having a weak match will lower the value). The matches with greatest cost will be returned.

I’m not sure what is meant by “infrastructure,” but I think DrRacket would be a good place to make a “toy” version of this idea.