High School Set Theory
Using Technology

Timothy W. Jones

October 21, 2022

Abstract

Simple set theory problems of finding unions, intersections, and complements of simple sets consisting of integers and single letters is not a trivial exercise to automate with technology. It can be done using TI-Basic programming in the case of sets consisting of positive integers. As a TI-84 doesn't have associative arrays or just arrays for that matter doing the same for single letters of the alphabet is just about impossible. But the task can be done in Python pretty elegantly. Why bother? Well the next evolution is to see the need and power of a database and its structured query language (SQL). This tie-in of basic set theory and databases seems to never be made. It is an important connection.

Introduction

Given the task of teaching set theory using a high school textbook I was thrown, I admit it, for a loop. My penchant while teaching algebra and other such topics was always to challenge myself and my classes to find an optimal use of technology. Can you get a calculator or spreadsheet to do all the pencil and eraser kind of work for you? So, for sure, you can implement the quadratic formula in a calculator or spreadsheet, input the three coefficients and viola you've done it. You can use various string manipulations to get a factored form, push a button and see a nice graph. But then what to do with the union of say $\{1, 2, 3\}$ with $\{8, 2, 3, 7, 10\}$ or $\{a, b, c\}$ with $\{u, v, a, r, c\}$?, the intersection of three such sets and other typical end of chapter problems? Why bother, one could protest, they're easy enough.

As I tried dredging out a solution with a TI-84 calculator a light bulb went off. It was the beginning of the semester and I was attempting to get students enrolled in the course to sign up for the homework. We use CANVAS (C) and myMathLab (M). The roster for the former is in effect the universal set and the one for myMathLab is a proper subset; the complement of M relative to C gives a list of those students I should send an email to and prod to sign up. I was doing set theory and what I really wanted was a single button to push that would make an inner join of two tables, pull out email addresses, combine all with a pre-written form letter and send the e-mails for me. Set theory is really the math behind databases.

Shocked at this revelation, I flipped through our textbook [1] and no mention was made of this fact. Far from being something you should not bother to automate, of all the topics in basic math, the one that should be connected up with automation is clearly set theory. We live in a world of databases and students should know a little bit about that, IMHO. Not to mention that in an office setting they might actually, really find lots of applications of set theory as enabled by Excel or even Access. Conceptual understanding is of value! Certainly I should keep going with my TI-84 attempts and see if I could do simple set theory problems.

In the remainder of this article I'll give a TI-84 solution to the problem. There are some limitations that Python solves readily. Excel has some easy features of interest. But the big final solution has to be SQL and wonder of wonder Python allows a fast import of SQLite. It works in Thonny (Python IDE). I'll touch on these.

Lists in a TI-84

TI-84 lists with some tricks does a lot. Even the nomenclature is perfect. You use curly braces and separate elements by commas. Another big feature is you can use arithmetic and logical operators on lists. If one confines oneself to lists with elements that are the natural numbers one can write programs that do unions, intersections, and complements. In fact, one can use the logic (off of the test menu) to perform these operations on the home page (i.e. without a program). Granted zeros must stand in for blanks and that is a little awkward; the sets are sorted.

In Figure 1, I have defined a universal set and set A using the build in lists L_1 and L_2. The *not* function used twice gives a binary array, in effect for set A. In Figure 2, set B is stored and its binary array is created; using these binary sets and the *or* operation (off the second test key) together with multiplication, the union is formed.

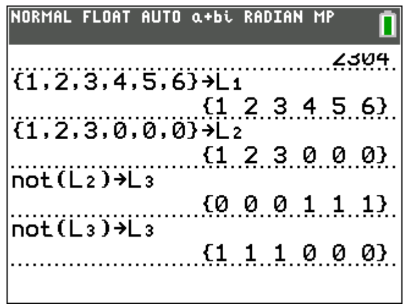


Figure 1: The universal set is stored in L_1 and A is stored in L_2.

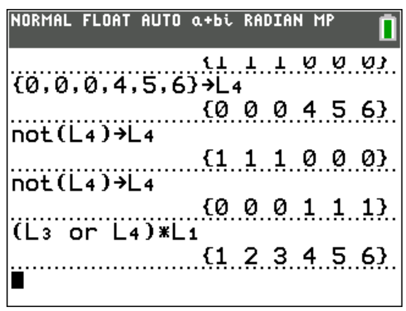


Figure 2: The set B is stored and the union of A and B is crunched.

One can see how these tricks will work for intersection: just change the *or* to an *and*. Switching to Connect from Smartview and coding intersection we can begin to evolve general solutions and start dreaming of drill down menus and maybe dropping the sorted requirement (not a slight) for our sets. Figures 3 and 4 show a program for intersection and its printout.

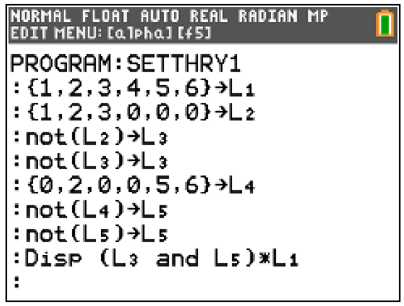


Figure 3: The build in lists are created using keys on the calculator – easy, fast.

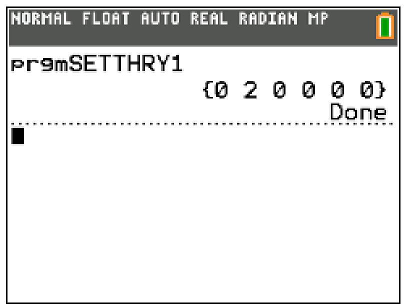


Figure 4: The intersection of two simple sets.

Adding menus

We can improve the interface. Let's stipulate that our universal set is always just $U = \{1, 2, 3, 4, 5, 6, 7, 8, 9, 10\}$. We want to just enter elements of sets A, B, and C without regard to order and without putting in placeholder zeros. Naturally, these sets must consist of just elements of U. We also want to add a menu system that allows us to not have to re-enter these sets, given we want to reuse previous definitions. We will show two drill down menus: one for entering sets and one for unions. This second menu is easily extended – one can add intersection, unions involving complements and other combinations. Of course, once the four sets are defined (U, A, B, and C) one can use the binary sets at will to do combinations from the home screen – see previous.

Speaking of binary sets, Figure 5 shows the central trick. We use the elements of list A to index its binary set, filling its appropriate slot with a 1. The top menu code is also shown in this figure.

VAR NAME: A

```
Menu("ENTER SETS","A",A,"B",B,"C",C,"U",U,"INSPECT",I,"REUSE",R,"NEXT",N,"QUIT",Q)
Lbl A
Prompt ʟA
10→dim(L1):Fill(0,L1)
For(X,1,dim(ʟA))
ʟA(X)→T
1→L1(T)
End
prgmA
```

Figure 5: All values in A are non-zero, so values index for proper placement.

The other coding challenge is to remove the zero placeholders. This is achieved by testing the value of the binary list. If it is a zero then we don't record the value. Figure 6 shows the idea. This clear-zeros program is called from the second menu. Figure 7 shows the portion of the code that gives $A+B$, A union B.

VAR NAME: CLRZEROS

```
0→Y
For(X,1,10)
If (L4(X))
Then
Y+1→Y
Y→dim(L5)
L4(X)→L5(Y)
End
End
Disp L5
```

Figure 6: This clears the zeros.

TI-84 Menu program

Here are screen captures, Figures 8 and 9, that give the look of the TI-84 solution to easy set theory problems.

VAR NAME: B

```
Menu("UNION(+)","A+B",A,"A+C",B,"B+C",C,"NEXT",N,"BACK",Q)
Lbl A
(L1 or L2)*ʟU→L4
prgmCLRZEROS
Pause
prgmB
```

Figure 7: This snippet gives A union B.

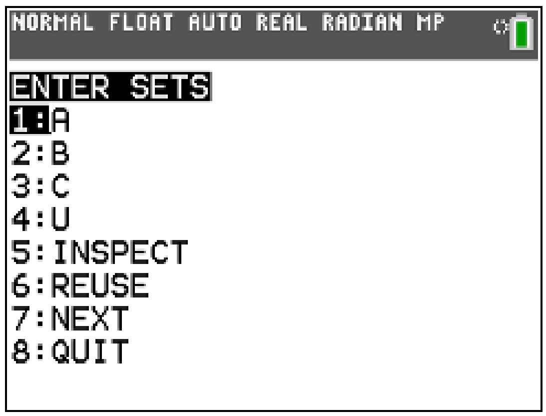


Figure 8: The menu gives options to inspect A, B, C, and U sets.

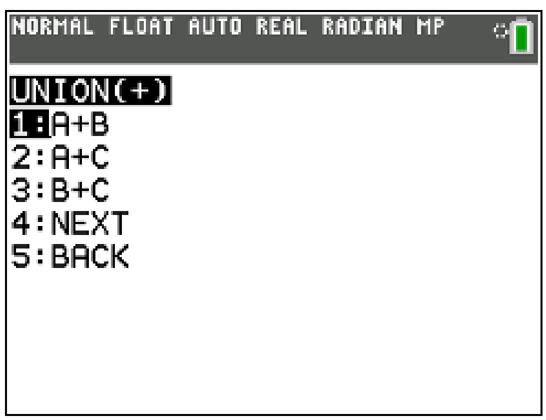
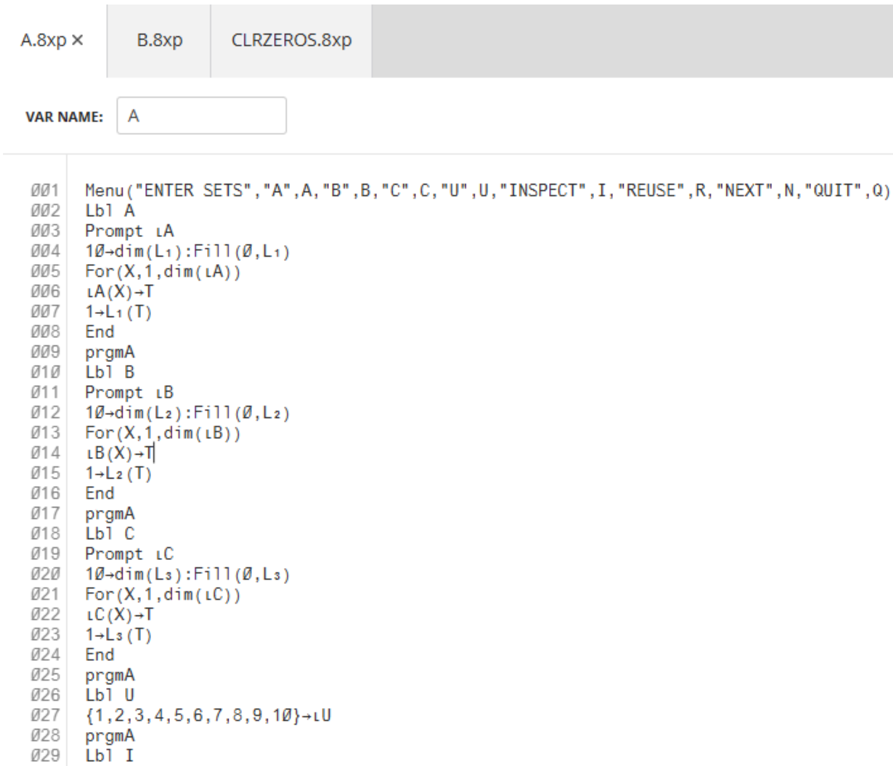


Figure 9: The second or drill down menu (from next on the main menu).

Complete Code

Here is the code for the menu system of the previous section – all of it.




```
Menu("ENTER SETS","A",A,"B",B,"C",C,"U",U,"INSPECT",I,"REUSE",R,"NEXT",N,"QUIT",Q)
Lbl A
Prompt ʟA
10→dim(L1):Fill(0,L1)
For(X,1,dim(ʟA))
ʟA(X)→T
1→L1(T)
End
prgmA
Lbl B
Prompt ʟB
10→dim(L2):Fill(0,L2)
For(X,1,dim(ʟB))
ʟB(X)→T
1→L2(T)
End
prgmA
Lbl C
Prompt ʟC
10→dim(L3):Fill(0,L3)
For(X,1,dim(ʟC))
ʟC(X)→T
1→L3(T)
End
prgmA
Lbl U
{1,2,3,4,5,6,7,8,9,10}→ʟU
prgmA
Lbl I
```

Figure 10: Caption for set-theory-hard-screen-1-through029

A.8xp × | B.8xp | CLRZEROS.8xp

VAR NAME: A

```
10→dim(L2):Fill(0,L2)
For(X,1,dim(ʟB))
ʟB(X)→T
1→L2(T)
End
prgmA
Lbl C
Prompt ʟC
10→dim(L3):Fill(0,L3)
For(X,1,dim(ʟC))
ʟC(X)→T
1→L3(T)
End
prgmA
Lbl U
{1,2,3,4,5,6,7,8,9,10}→ʟU
prgmA
Lbl I
ClrHome
Disp "A=L1",L1
Disp "B=L2",L2
Disp "C=L3",L3
Disp "U=",ʟU
Pause
prgmA
Lbl R
prgmB
Lbl N
prgmB
Lbl Q
Stop
```

Figure 11: Caption for set-theory-hard-screen-1-013-042

A.8xp | B.8xp × | CLRZEROS.8xp

VAR NAME: B

```
Menu("UNION(+)","A+B",A,"A+C",B,"B+C",C,"NEXT",N,"BACK",Q)
Lbl A
(L1 or L2)*ʟU→L4
prgmCLRZEROS
Pause
prgmB
Lbl B
(L1 or L3)*ʟU→L4
prgmCLRZEROS
Pause
prgmB
Lbl C
(L2 or L3)*ʟU→L4
prgmCLRZEROS
Pause
prgmB
Lbl N
Disp "N"
Pause
Lbl Q
prgmA
```

Figure 12: Caption for set-theory-hard-screen-2-all-of-it

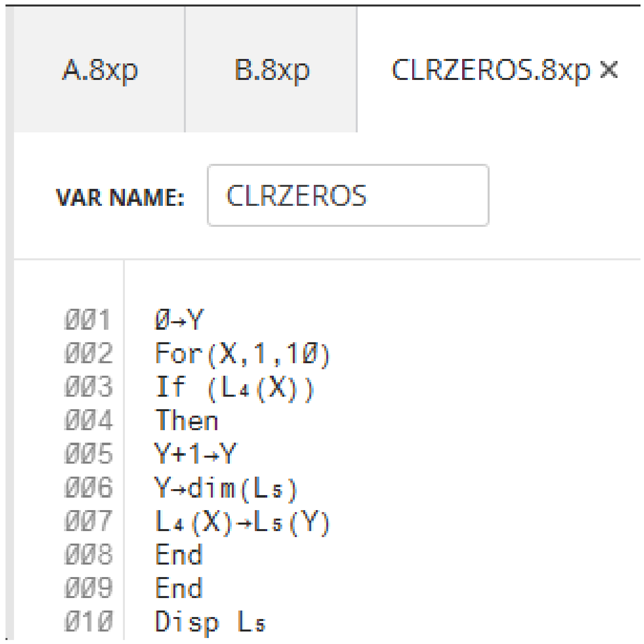


Figure 13: Caption for set-theory-hard-screen-3-all-of-it

Other technology

The TI84 solution is limited to natural numbers. Typically textbooks use natural numbers, but they also use letters for simple problems. A TI84 does have 9 built in string variables, but these are not in the form of an array. Matrices and lists only support number elements, so there really is no way to represent lists comprised of letters.

Python and all other major programming languages, of course, will support arrays of strings. As the latest TI84-CE with Python has Python, you might wonder how difficult it is to create the equivalent of the TI-Basic program just given. Surprisingly, there are some annoyances. Strings in arrays must be quoted. In TI-Basic no quotes (numbers don't require quotes). The logical operators are not overloaded, in effect, like they are in TI-Basic. The import *numpy* gives this functionality, albeit not as elegantly as one (I) would like.

C++ with its ability to define operators might be able to do union and intersection with symbols – of course the common keyboard doesn't come with union and intersection or not. A palette idea might be of interest. Thence to a webpage and javascript, maybe regular expressions.

Excel has a easily used filter button on a ribbon that places drop down arrows on headers for tables. These drop downs allow for some set theory functions to be implemented fast. The spill phenomenon can make these tables annoying for real-word, repetitive use.

Here it is: structured query language allows for complicated unions, intersections, and all possible queries. SQL thus can and should be introduced in an intro to set theory. Ideally, an easy query giving M', those who haven't signed up for homework would be crunched out right before the students eyes. It's an inner join with a not.

Conclusion

Here's a capture from a youtube video showing the power of SQL.

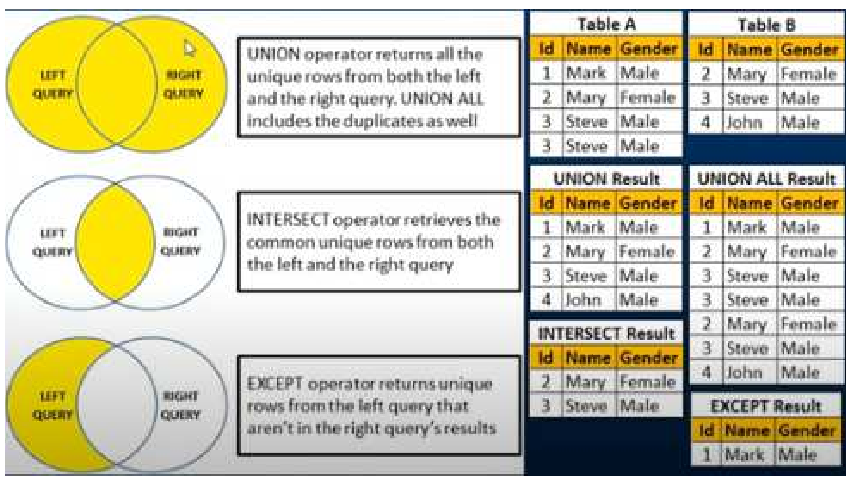


Figure 14: PragrimTech.com by permission.

References

[1] Blitzer, R. (2022). *Thinking Mathematically*, 8th ed. Hoboken, NJ: Pearson.