BRAID DATA CLEAN-UP PSEUDOCODE

JESSE HAMER

The following is a pseudocode algorithm for the program used to clean up quasi-positive and strongly quasipositive braid data from KnotInfo.

The code was written using Python 2.7 in the Jupyter notebook environment. In particular, it makes heavy use of the package *openpyxl* to manipulate Excel files, as well as the *Sage* package to do all computations involving braid words, links, and Jones polynomials.

Each section will describe a method used in the algorithm, with the final section being the main "braid_clean" algorithm. The section title will list the expected inputs of each method, and the section will begin with a short description of the purpose of the method. Statements preceded by '#' denote comments

All methods assume that braid representatives from the table have index ¡ =10 (which is true).

1. EXTRACT_BN(CELL)

This function expects as input a unicode string comma separated list of numbers surrounded by a pair of braces. It takes this unicode string, strips it of all braces and commas, and builds a Python list structure containing only the list of numbers representing a braid word. *extract_bn* is not built to handle knots whose braid notation entry contains two representatives; see "extract_single_bn" instead.

```
initialize b as the empty list []
while i < length of cell:
    if(the ith member of cell is a curly brace, comma, or white space)
        i = i+1
    else if (the ith member of cell is numeric)
        append the ith member of cell to b
        i = i + 1
    else if (the ith member of cell is - and the (i+1) member of cell is numeric)
        append the ith and (i+1)th members of cell to b
        i = i+2
    else
        return the empty list       #The ith member of cell is an unexpected character
return b
```

Date: October 26, 2017.

2. EXTRACT_SINGLE_BN(CELL)

This is nearly the same algorithm as was used for *extract_bn*. The only differences are:

(a) The expected input is a pair of unicode string comma separated lists of numbers surrounded by curly braces, which are themselves surrounded by curly braces: {{comma sep. list of numbers},{comma sep. list of numbers}}
(b) When the first right curly brace '}' is read, we stop reading and return the numeric list which has been built so far.

3. CHECK_MIRROR(L_1,L_2,INDEX,ERRORS)

This function will check whether the given links L_1 and L_2 are mirror images by comparing their Jones' polynomials. The index of the row in the table from which the links L_1 and L_2 were derived is passed, as well as a list of mismatch errors which is to be updated with index in the event that L_1 and L_2 are found to be neither equal nor mirror images. The function returns the string 'Y' if L_1 and L_2 are found to be mirrors, 'N' if they are equal, and 'uncertain' if either L_1 or L_2 is amphichiral with respect to the Jones polynomial (the Jones polynomial of the link and its mirror are the same) or if there is a mismatch error.

```
if(the Jones polynomials of L_1 and L_2 are the same):
    if(the Jones polynomial L_1 is equal to that of its mirror,
        or the Jones Polynomial of L_2 is equal to that of its mirror):
        print that there is an error in row 'index'
        return 'uncertain'
    else:
        return 'N'
if(the Jones polynomial of L_1 is equal to that of the mirror of L_2):
    if(the Jones polynomial L_1 is equal to that of its mirror,
        or the Jones Polynomial of L_2 is equal to that of its mirror):
        print that there is an error in row 'index'
        return 'uncertain'
    else:
        return 'Y'
else:
    append index to errors   #There is a mismatch
```

4. RADIUS_OF_CONJ(QPBRAID,PIVOT,ENDS_OF_BANDS)

A quasipositive band generator is one of the form $b\sigma_i b^{-1}$, where σ_i is a generator in the braid group, and b is any braid word. This function takes a numeric list representing a quasipositive braid word, 'qpbraid', and an index 'pivot' which represents a letter σ_i in a supposed quasipositive band generator of 'qpbraid'. The function returns the maximal radius r about 'pivot' so that the subword of 'qpbraid' from 'pivot' - r to 'pivot' + r is a quasipositive band generator. We also pass a numeric list 'ends_of_bands' whose elements are the rightmost ends of quasipositive (or strongly quasipositive) braids already found in 'qpbraid' to

the left of 'pivot'. This is used to ensure that we do not accidentally have overlapping band generators.

```
r = 0
while (pivot - r - 1 is at least 0,
    and pivot +r + 1 is smaller than the length of qpbraid,
    and qpbraid[pivot -r - 1] equals -qpbraid[pivot + r + 1],
    and qpbraid[pivot -r - 1] is not in ends_of_bands):
        r = r+1
return r
```

5. INVERT_BAND(BAND)

This function takes a subword 'band' of a braid word and computes its inverse as an element of the braid group. It returns a numeric list representing the inverse.

```
inverse = empty list
i = 0
while (I < length of band):
    append -band[len(band) - 1 - i] to inverse
    i = i+1    #walk backwards through band, negating and appending to inverse
return inverse
```

6. SEARCH_LIST(L,S,START)

This function searches through the list 'l' for the substring 's', starting at the index 'start'. It returns a list of indices where all instances of the substring 's' in 'l' begin, or 0 if no such substring is found in 'l'.

```
k = length of s
i = start
indices = empty list
found = FALSE
while (i < length of l):
    if(the slice of l from i to i +k is equal to s):
        found = TRUE
        i = i+1
    else:
        i = i+1
if (found):
    return indices
else:
    return 0
```

7. format_pbraid(is_mirror, pbraid)

This function takes a numeric list 'pbraid' representing a positive braid word and a string 'is_mirror' which is the output of the function *check_mirror* from above, and returns a formatted unicode string ready to be placed on the cleaned spreadsheet. The formatting for a positive braid simply surrounds the positive braid word in a pair of curly braces, with an 'M' preceding the first curly brace if pbraid is the mirror image of the original braid.

```
formatted = empty unicode string
i = 0
l = length of pbraid
if(is_mirror is 'Y'):
    append 'M{' to formatted
else:
    append '{' to formatted
while (i < l):
    if(i is 0):
        append the first element of pbraid to formatted
        i = i+1
    else:
        append a comma and then the ith element of pbraid to formatted
        i = i+1
append '}' to formatted
```

8. format_sqpbraid(is_mirror, sqpbraid)

This function takes a numeric list 'sqpbraid' representing a strongly quasipositive braid word and a string 'is_mirror' which is the output of the function *check_mirror* from above, and returns a formatted unicode string ready to be placed on the cleaned spreadsheet. The formatting for a strongly quasipositive braid surrounds the entire braid word and any strongly quasipositive band generators in a pair of curly braces, with an 'M' preceding the first curly brace if sqpbraid is the mirror image of the original braid.

```
formatted = empty unicode string
i = 0
l = length of sqpbraid
if(is_mirror is 'Y'):
    append 'M{' to formatted
else:
    append '{' to formatted
while(i < l):
    k = 0
    if(i is 0):
        append sqpbraid[i] to formatted
        i = i+1
    else:
        if(sqpbraid[i] is negative): #we are in a sqp band
```

```
            while(i + k < l and sqpbraid[i+k] <  0):
                k = k+1 #find length of sqp band
            insert a '{' left of the element of formatted corresponding to sqpbraid[i-k]
            for j between 0 and k:
                append sqpbraid[i + j] to formatted
            append a '}' to formatted
            i = i +k
        else:
            append sqpbraid[i] preceded by a comma to formatted
            i = i+1
append '}' to formatted
return formatted
```

9. FORMAT_QPBRAID(IS_MIRROR, QPBRAID)

This function takes a numeric list 'qpbraid' representing a quasipositive braid word and a string 'is_mirror' which is the output of the function *check_mirror* from above, and returns a formatted unicode string ready to be placed on the cleaned spreadsheet. The formatting for a quasipositive braid surrounds the entire braid word and any quasipositive band generators in a pair of curly braces, with an 'M' preceding the first curly brace if sqpbraid is the mirror image of the original braid. If a quasipositive band generator contains within it nested quasipositive bands, then braces are only placed around the enclosing quasipositive band generator.

```
formatted = empty unicode string
i = 0
l = length of qpbraid
ends_of_bands = empty list    #will keep track of right-ends of qp bands
if(is_mirror is 'Y'):
    append 'M{' to formatted
else:
    append '{' to formatted
while (i < l):
    if (i is 0): #make sure not to precede first element with a comma
        if(qpbraid[i] is negative): #we are in a quasipositive band
            k = 0
            pivot = i +1 #will look for the pivot; use i +1 as a first guess
            found = 'N'
            whlie(found is 'N'):
                if(pivot < l and qpbraid[pivot] > 0): #possible pivot; must check
                    k = radius_of_conj(qpbraid, pivot, ends_of_bands)
                    if (k is at least pivot - i): #found qp band containing index i
                        #must now find the maximal qp band containing i
                        test_indices = search_list(qpbraid,
                            invert_band(slice of qpbraid from pivot -k to pivot +k),
                            pivot + k +1)
                        if(test_indices is nonempty):
                            for j in test_indices:
```

```
                                test_pivot = (j + pivot + k)/2
                                if(test_pivot is an integer and
                                    radius_of_conj(qpbraid, test_pivot, ends_of_bands)
                                    is large enough to contain original qp band):
                                        k = radius_of_conj(qpbraid, test_pivot,
                                            ends_of_bands)
                                        pivot = test_pivot
                            found = 'Y'
                        else: #keep looking to the right, pivot not found
                            pivot = pivot + 1
                    else:
                        pivot = pivot +1
                        if(pivot > l):
                            return the empty string
                            #error: negative generator is not in a qp band
                insert '{' before member of formatted corresponding to beginning of new qp band
                append qpbraid[i] to formatted
                for j from i+1 to pivot +k + 1:
                    append qpbraid[j] preceded by a comma to formatted
                append '}' to formatted
                append pivot + k to ends_of_bands
                i = pivot + k +1
            else:
                append qpbraid[i] to formatted
                i = i+1
        else: #i is not zero, now we must precede all entries by commas
            if(qpbraid[i] is negative):
                #first test if we're in a sqp band
                k = 0
                if(i-1 is not in ends_of_bands):
                    k = radius_of_conj(qpbraid, i-1,ends_of_bands)
                if(k > 0): #we're in a sqp band
                    pivot = i-1
                    test_indices = search_list(qpbraid,
                                invert_band(slice of qpbraid from pivot -k to pivot +k),
                                pivot + k +1)
                    if(test_indices is nonempty):
                        for j in test_indices:
                            test_pivot = (j+pivot +k)/2
                            if(test_pivot is an integer and test_pivot is the pivot
                                    of a qp band which contains the original sqp band):
                                #we are in a nested qp band; find maximal such band
                                k = radius_of_conj(qpbraid, test_pivot,ends_of_bands)
                                pivot = test_pivot
                    if(pivot - k is 0):
                        insert '{' before the member of formatted corresponding to qpbraid[0]
                        for j in i to pivot + k:
```

```
            append qpbraid[j] preceded by a comma to formatted
    else:
        insert ‘,{’ before the member of formatted corresponding to qpbraid[0]
        for j in i to pivot + k:
            append qpbraid[j] preceded by a comma to formatted
    append ‘}’ to formatted
    append pivot + k to ends_of_bands
    i = pivot + k +1
else:
#not in sqp band; must be in qp band. Take i +1 as first guess for pivot.
k = 0
pivot = i+1
found = ’N’
while(found is ‘N’):
    if(pivot < l and qpbraid[pivot] > 0):
        k = radius_of_conj(qpbraid, pivot, ends_of_bands)
        if(k is at least pivot - i):
            #we’re in a qp band containing original index i
            test_indices = search_list(qpbraid,
                    invert_band(slice of qpbraid from pivot -k to pivot +k),
                    pivot + k +1)
            if(test_indices is nonempty):
                for j in test_indices:
                    test_pivot = (j+pivot +k)/2
                    if(test_pivot is an integer and test_pivot is the pivot
                        of a qp band which contains the original sqp band):
                        #we are in a nested qp band; find maximal such band
                        k = radius_of_conj(qpbraid, test_pivot,ends_of_bands)
                        pivot = test_pivot
            found = ‘Y’
        else:
            #keep looking right
            pivot = pivot +1
    else:
        pivot = pivot +1
        if(pivot > l):
            return empty string #error: qpbraid is not in qp form
if(pivot - k is 0):
    insert ‘{’ before the member of formatted corresponding to qpbraid[0]
    for j in i to pivot + k:
        append qpbraid[j] preceded by a comma to formatted
else:
    insert ‘,{’ before the member of formatted corresponding to qpbraid[0]
    for j in i to pivot + k:
        append qpbraid[j] preceded by a comma to formatted
append ‘}’ to formatted
append pivot + k to ends_of_bands
```

```
            i = pivot + k +1
        else:
            append qpbraid[i] preceded by a comma to formatted
            i = i+1
append '}' to formatted
return formatted
```

10. CLEAN_OUTPUT(ROW, IS_MIRROR,PBRAID,SQPBRAID,QPBRAID)

This function takes an entire row, 'row', of the spreadsheet along with the output of *check_mirror*, 'is_mirror', and (possibly empty) numeric lists representing positive ('pbraid'), strongly quasipositive ('sqpbraid'), or quasipositive ('qpbraid') braid words for the data contained in row. It will call the formatting functions defined above, and place the output back on to the spreadsheet. Moreover, the precedence of positivity is respected: if, for example, the given braid has a positive word, then this positive word will be placed in the positive notation, strongly quasipositive notation, and quasipositive notation columns of the given row. In the quasipositive case, if an empty string was returned by format_qpbraid then an error message indicating an out of bounds exception is reported.

```
if(pbraid is nonempty):
    temp = format_pbraid(is_mirror, pbraid)
    set positive, sqp, and qp notation columns of row as temp
else if (sqpbraid is nonempty):
    temp = format_sqpbraid(is_mirror, pbraid)
    set sqp and qp notation columns of row as temp
else if (qpbraid is nonempty):
    temp = format_qpbraid(is_mirror, qpbraid)
    if(temp is nonempty):
        set qp notation column of row as temp.
    else:
        print("Out of bounds error in row")
```

11. MAIN FUNCTION: BRAID_CLEAN

This is the main function to clean the braid data and save the results to a new spreadsheet. A row counter is maintained so that any error messages can be accurately reported. A list of rows with mismatched braid and qp/sqp notation is generated and printed at the end. Any unexpected notation from the original "dirty" data is caught and an error message is printed. Positive, strongly quasipositive, and quasipositive braid notations are formatted according to the corresponding methods described above. If the braid notation column of a row contains two representatives, the first is used for mirror-checking purposes, and then the pair is formatted as a pair of braced, comma-separated, numeric lists, surrounded by brackets. Also, any mismatches of the following kind are caught and reported: for example, the quasipositive column says 'N', but the quasipositive notation column says 'existence unknown'. The user has the options to clean or not clean the data as well as save or not save the data. If the data is not cleaned, only checks for mismatches will be performed.

```
Load openpyxl, as well as Link and BraidGroup from Sage
row_counter = 4 #first row on which a nontrivial braid occurs
mismatches = empty list
ask user if they want to clean the data and save this answer to CLEAN
use openpyxl to load the dirty braid data as a workbook; call it wb
set ws to be the active worksheet of wb
set B to be the 10-stranded braid group

#iterate through the rows of ws
for row in ws (starting at row 4):
#extract data from the row
    name = row[0]
    nr = row[1]
    bn = row[2]
    pb = row[3]
    p = row[4]
    sqp = row[5]
    qp = row[6]
    pn = row[7]
    ppdn = row[8]
    sqpn = row[9]
    qpn = row[10]

    braid = empty list
    pbraid = empty list
    sqpbraid = empty list
    qpbraid = empty list

    #now we extract Link objects, or return errors if this is not possible
        and we have bad data
    if(first two members of bn are ‘{{’):
        braid = extract_single_bn(braid)
    else:
        braid = extract_bn(bn)
    if(braid is empty):
        throw a bad data entry error
    else:
        L = Link(B(braid))
        #B() creates a braid object from a numeric list; Link creates a
            link object from a braid.
    if(L has more than one component):
        throw error; we should not have knots
    if(pb == ‘Y’ and pn is not ‘exists’):
        pbraid = extract_bn(pn)
        if(pbraid is empty):
            throw a bad data entry error
        else:
```

```
            Lp = Link(B(pbraid))
    else if (pb is not ‘N’ or pn is not ‘does not exist’,
        and pb is not ‘unknown’ or pbn is not ‘existence unknown’):
        throw a bad data entry error
    else if (sqp is ‘Y’ and sqpn is not ‘exists’):
        sqpbraid = extract_bn(sqpn)
        if(sqpbraid is empty):
            throw a bad data entry error
        else:
            Lsqp = Link(B(sqpbraid))
    else if (sqp is not ‘N’ or sqpn is not ‘does not exist’,
        and sqp is not ‘unknown’ or sqpn is not ‘existence unknown’):
        throw a bad data entry error
    else if (qp is ‘Y’ and qpn is not ‘exists’):
        qpbraid = extract_bn(qpn)
        if(qpbraid is empty):
            throw a bad data entry error
        else:
            Lqp = Link(B(qpbraid))
    else if (qp is not ‘N’ or qpn is not ‘does not exist’,
        and qp is not ‘unknown’ or qpn is not ‘existence unknown’):
        throw a bad data entry error

    #now check whether ay positive/qp/sqp entry is the mirror of the given braid rep.
    if(pbraid is nonempty):
        is_mirror = check_mirror(L,Lp, row_counter,mismatches)
    else if(sqpbraid is nonempty):
        is_mirror = check_mirror(L,Lsqp, row_counter,mismatches)
    else if(qpbraid is nonempty):
        is_mirror = check_mirror(L,Lqp, row_counter,mismatches)

    if(CLEAN is ‘Y’):
        if(bn starts with ‘{{’ and ends with ‘}}’):
            replace initial and terminal curly braces with brackets
        clean_output(row, is_mirror, pbraid,sqpbraid,qpbraid)

    row_counter = row_counter +1

ask user if they want to save the data, store this answer as SAVE
if(SAVE is ‘Y’):
    ask user for a file name to save to; store this answer as SAVE_NAME
    save the cleaned workbook as SAVE_NAME.xlsx
    #warning: this overwrites any file previously named SAVE_NAME.xlsx
close the workbook
print mismatches
```