GST Standard Input Formats

May 30, 2014

1 Introduction

This document is intended to be a reference describing in detail the standard syntax expected of GST input files. The level of detail in this document is more than the average customer of GST will need.

2 Gate Strings

The names of gates can be any string beginning with "G" (case-sensitive!) and followed by any number of *lowercase* alphabet characters, numerals 0 through 9, or the underscore character. A gate string is a sequence of gate names. The names don't have to be separated by anything, since a capital G marks a new gate name, but they can be separated by whitespace or an asterisk (*) character. Parentheses can be used in pairs where convenient. To repeat a gate or parenthesized string of gates the caret (^) followed by an integer can be used. See the examples below which demonstrate this usage. The empty gate string indicating that no gates were performed is a special case, and is denoted using the special symbol {}.

3 Input file formats

There are four types of input files. Below we list their descriptions followed by an example of each. A syntax common to all three types is that any line beginning with a pound sign (#) and followed by any alphanumeric character or whitespace is considered a comment and ignored. This is illustrated in many of the examples which follow.

- Gate string list files: just lists a sequence of gate strings (no counts, just strings). Each line contains a single gate string.

```
G1
G1G2
G2 G3
G1*G3
GxGcoolstuff^3
G1(G1G3)^2
```

Listing 1: Example gate string list file

- Gate string dictionary files: associates (potentially long) gate strings with a (usually sort) label. Each line contains a label name (no spaces are allowed, but underscores are), followed by a space, followed by a gate string.

```
#My Dictionary file
1 G1
2 G1G2
3 G1G2G3G4G5G6
MyFav1 G1G1G1
MyFav2 G2^3
this1  G3*G3*G3
thatOne G1 G2 * G3
```

Listing 2: Example gate string dictionary file

- Data files: associates gate strings with data counts. Optionally one can include "preamble directives" at the beginning of the file by specifying lines that begin with "##" before the first non-comment line of the file. Available directives include:

– Lookup = *dictionary_file_name* – allows one to use the labels defined in the dictionary file (to make the data file more readable)

– Columns = *comma_separated_column_names* – identifies the columns that follow the gate string. The default is "`plus frequency, count total`", and other available column names are `minus frequency`, `plus count`, and `minus count`.

```
#My Data file
# Get string lookup data from the file test.dict, and specify that the columns
#    following the gate sequences are the plus (+) outcome frequency and the
#     total number of outcomes
## Lookup  = test.dict
## Columns = plus frequency, count total

#empty string
{}            1.0 100

#simple sequences
G1G2          0.098  100
G2 G3         0.2    100
(G1)^4        0.1   1000

#using lookups
G1 S[1]       0.9999 100
S[MyFav1]G2   0.23   100
G1S[2]^2      0.5     20
S[3][0:4]     0.2      5
G1G2G3G4      0.2      5

#different ways to concatenate gates
G_my_xG_my_y  0.5 24.0
G_my_x*G_my_y 0.5 24.0
G_my_x G_my_y 0.5 24.0
```

Listing 3: Example data file

- Gate set files: specifies a set of gates and state preparation and measurement operations. Each object (gate, state prep, or measurement) is given by a block of lines containing no blank lines, and blocks are separated by blank lines. Each block begins has the following format:

  1. first line: a label for the object. In the case of state preparation and measurements, this can be any name you choose but the default names of "rho" and "E" must be used for preparation an measurement respectively if you don't specify any spam label (see below). Gate labels should be a valid gate names (so should begin with a "G" and followed by lowercase characters, numerals, or the underscore character as described above).
  2. second line: a format name, specifying what the format of the following data. Allowed formats are:
     – StateVec : prep/measure action as a quantum state vector in the standard sigma-z basis
     – DensityMx : prep/measure action as a density matrix in the standard sigma-z basis
     – PauliVec : prep/measure action as a density matrix vector in the Pauli basis
     – UnitaryMx : gate action as a unitary matrix
     – UnitaryMxExp : gate action as a hermitian matrix that when multiplied by $-i$ and exponentiated gives the unitary gate action.
     – PauliMx: gate action as a matrix in the Pauli basis.
  3. third line until end of block: rows of space-delimited data elements. A vector block contains only a single row of data, wherease matrix blocks contain multiple data rows. Each element must evaluate to a real or complex number depending on the context (e.g. PauliMx elements must be real wheres UnitaryMx elements can be complex). Evaluation is done via the python interpreter, so the imaginary $\sqrt{-1}$ is given by a lowercase j. The symbols `sqrt` and `pi` are available for use in evaluations. For example, `5`, `5+4j`, `0.1*pi`, and `2*(0.3+1j*pi)/sqrt(2)` are all valid element expressions (just note there can be no spaces within the element expression, as this is used to delimit the elements).

In addition to object blocks, a gate set file may contain lines which specify spam labels. Such lines associate a name with a (state prep, state measurement) pair, and are used to name the outcomes of different experiments. One can also

specify a label to be associated with the remaining outcomes not given by other spam labels. The format of such lines is:
SPAMLABEL *labelname* = *state_prep_label measurement_label*
or SPAMLABEL *labelname* = remainder
If no spam labels are specified, the default is to create a "plus" label that corresponds to using the "rho" state preparation and the "E" measurement (so you must use the labels "rho" and "E" for this to work), and "minus" label that corresponds to all remaining outcomes. Below are several examples of gateset files which all specify the same gate set in different ways.

```
# My gateset

#State prepared, specified as a density matrix
rho
DensityMx
1 0
0 0

#State measured, specified as a density matrix
E
DensityMx
0 0
0 1

#First gate, specified as a unitary matrix:  X(pi)
G1
UnitaryMx
0 1
1 0

#Second gate, specified as a unitary matrix: X(pi/2)
G2
UnitaryMx
 1/sqrt(2)   -1j/sqrt(2)
-1j/sqrt(2)   1/sqrt(2)
```

Listing 4: First example gate set file

```
# My gateset again

#State prepared, specified as a state
rho
StateVec
1 0

#State measured as "yes" outcome, specified as a state
E
StateVec
0 1

#First gate specified as H such that the unitary gate is U = exp(-i*H):  X(pi)
G1
UnitaryMxExp
0    pi/2
pi/2    0

#Second gate, specified as H such that the unitary gate is U = exp(-i*H): X(pi/2)
G2
UnitaryMxExp
0     pi/4
pi/4     0
```

Listing 5: Second example gate set file

```
# My gateset yet again

```

```
#State prepared, specified as a vector in the Pauli basis
myrho
PauliVec
1.0/sqrt(2) 0 0 1.0/sqrt(2)

#State measured, specified as a vector in the Pauli basis
myE
PauliVec
1.0/sqrt(2) 0 0 -1.0/sqrt(2)

#First gate specified as as matrix in the Pauli basis: X(pi)
G1
PauliMx
1 0  0  0
0 1  0  0
0 0 -1  0
0 0  0 -1

#Second gate specified as matrix in the Pauli basis: X(pi/2)
G2
PauliMx
1 0 0  0
0 1 0  0
0 0 0 -1
0 0 1  0

#Must specify spam labels since I changed the state prep and
# measure labels from the standard rho and E labels
SPAMLABEL plus = myrho myE
SPAMLABEL minus = remainder
```

Listing 6: Third example gate set file

Here is the Python Dataset dictionary object that results from reading the above data file (note that the dataset combines the counts of identical strings so that each key is unique):

```
{
()  :  {'plus': 100.0, 'minus': 0.0}
('G1', 'G2')  :  {'plus': 9.8, 'minus': 90.2}
('G2', 'G3')  :  {'plus': 20.0, 'minus': 80.0}
('G1', 'G1', 'G1', 'G1')  :  {'plus': 100.0, 'minus': 900.0}
('G1', 'G1')  :  {'plus': 99.99, 'minus': 0.01}
('G1', 'G1', 'G1', 'G2')  :  {'plus': 23.0, 'minus': 77.0}
('G1', 'G1', 'G2', 'G1', 'G2')  :  {'plus': 10.0, 'minus': 10.0}
('G1', 'G2', 'G3', 'G4')  :  {'plus': 2.0, 'minus': 8.0}
('G_my_x', 'G_my_y')  :  {'plus': 36.0, 'minus': 36.0}
}
```

Listing 7: Resulting internal dictionary resulting from the example data file above

4 The Grammar

The actual grammar, for those who care, is:

```
 nop     :: '{}'
 gate    :: 'G' [ lowercase | digit | '_' ]+
 reflbl  :: (alpha | digit | '_')+
 strref  :: 'S' '[' reflbl ']'
 slcref  :: strref [ '[' integer ':' integer ']' ]
 expable :: gate | slcref | '(' string ')' | nop
 expdstr :: expable [ '^' integer ]*
 real    :: ['+'|'-'] integer [ '.' integer [ 'e' ['+'|'-'] integer ] ]

 string  :: expdstr [ [ '*' ] expdstr ]*
```

```
dictline :: reflbl string
dataline :: string [ real ]+
```

Listing 8: The formal grammar used to parse GST input files

The final three grammar types correspond to the three types of input files:

- Gate string list files: each non-comment or blank line = string
- Gate string dictionary files: each non-comment or blank line = dictline
- Data files: each non-comment or blank line = dataline

Additionally, a data file can have a preamble before any data containing lines that specifies directives. Each preamble line is of the form "## key = value", and in this way one can specify a dictionary file to associate with a data file. Each type of file can contain blank lines and comments, which are indicated by a "#" at the beginning of the line. However, there can be no blank lines before the preamble of a data file.