2Knots User's Guide (C++ version)

Jacob Towber
DePaul University

Glenn Lancaster
DePaul University

December 03, 2004

Contents

1 Overview

A set of C++ classes is provided to support computation of certain isotopy invariants related to subsets of the Carter-Rieger-Saito movie moves in the theory of smoothly knotted surfaces. These knots are described in J. Scott Carter and Masahico Saito, *Knotted Surfaces and Their Diagrams American Mathematical Society*, Providence, 1998.

The isotopy invariants are described more fully in [LLT].

We first describe in Section 2 the major C++ classes used to represent knots as *movies*. In the Section 4 we specify the input format of movies required by the application programs. A brief description of these application programs and how to run follows in Section 6. Finally, some example outputs are given in Section 7.

2 Major C++ Classes

Smoothly knotted surfaces and more general 2-tangles are represented by *movies*. C++ classes are provided to implement the components of such movies. The Carter-Rieger-Saito movie moves are then expressed in terms of relations between these movies.

The major classes used to describe a movie are described here.

2.1 FEvent

The `FEvent` class represents a *framed event*. An **event** is one of {Cap,Cup,NE,NW}, which are represented pictorially in figure 1, where the arrow gives an orientation.

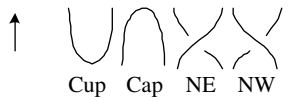

Figure 1: The 4 Elementary Events

A framed event consists of either a ordered triple of the form $[m, E, n]$ where m and n are natural numbers and E is an elementary event, **or** a symbol of

the form 1_n where n is a natural number. An example of the first kind of *framed event* is shown in figure 2 and is represented textually as [3,Cap,1] and figure 3 shows an example of the second kind.

Figure 2: The Framed Event [3,Cap,1]

Figure 3: The Framed Event 1_3

2.2 Still

The `Still` class represents a *still*. A still is a sequence of framed events with the property that the number of strings coming out of each framed event matches the number of strings going in to the next framed event. Figure 4 shows the still which is denoted by the sequence of 3 framed events: [2,NW,0][1,Cap,1][0,NE,0]. Note that in our conventions, this notation read from left to right corresponds to reading the diagram from **bottom** to **top**.

Figure 4: The still [2,NW,0][1,Cap,1][0,NE,0]

2.3 ET - Elementary Transitions

The class `ET` represents a general *elemntary transition*. In [CRS] these are referred to as *fundamental elementary string interactions*. Each elementary transition $\mathcal{E}$ is an ordered pair of stills, of which the first is called the **source** of $\mathcal{E}$, and the second the **target** of $\mathcal{E}$. An elementary transition $\mathcal{E}$ is uniquely determined by its source S and target T, in which case we shall write

$$\mathcal{E} = [S \Longrightarrow T] \,. \tag{1}$$

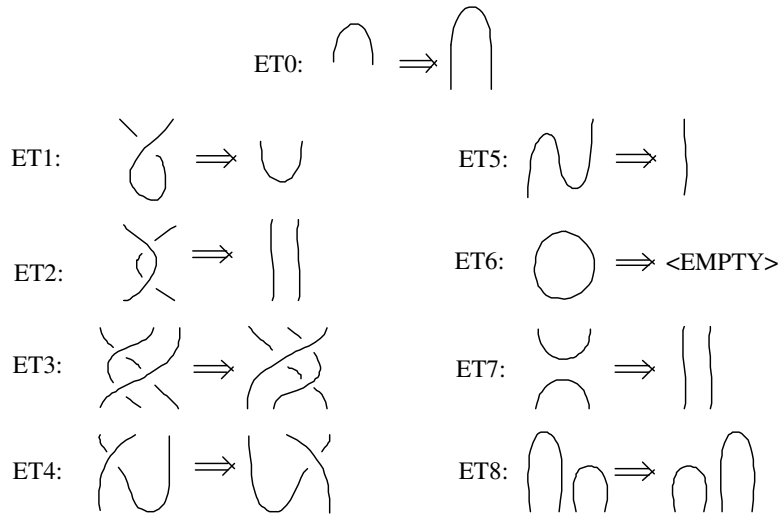

Figure 5: Elementary Transitions

The elementary transitions fall into 9 types, samples of which are pictured in Fig. 5.

2.4 Flicker

The C++ `Flicker` class represents a *flicker*.

A *flicker* is obtained from an elementary transition $\mathcal{E} : U \Longrightarrow V$ upon enhancing $\mathcal{E}$ by a 'bottom still' B below, m vertical strings to the left of $\mathcal{E}$ and n to the right, and then a 'top still' T above. A flicker can be represented by a pair of stills augmented to show the elmentary transition components, U and V.

An example flicker shown in Figure 6 is given textually by

```
1_3s[2,Cup,1]s[2,Cap,1][1,NW,0] => 1_3f[2,Cup,1][2,NE,1]f[2,Cap,1[1,NW,0]
```

The **sf** notation augments the stills to indicate the source and target of the elementary transition. The still enclosed in the **s**'s denotes the source still of the enhanced elementary transition, while the still enclosed by the **f**'s denotes the target. The bottom and top are common to both stills.

The flicker *source* is defined to be the bottom still and flicker *target* is defined as the top still. In this example, the bottom still is 1_3 and the top still is $[2, Cup, 1][2, NE, 1]f[2, Cap, 1[1, NW, 0]$.
Note that in this textual notation, reading from left to right corresponds to following the diagram from the bottom to the top.

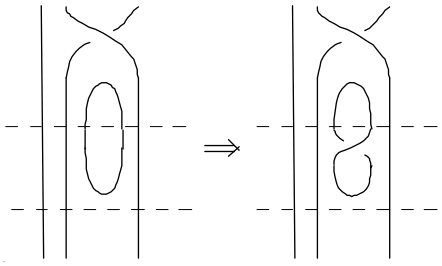

Figure 6: Example Flicker

2.5 Movie

A Carter-Rieger-Saito *m*ovie) is a finite non-empty sequence of flickers such that the target of each flicker in the sequence is equal to the target of the next flicker. The C++ `Movie` class represents such a movie.

Here is an example movie:

```
[0,Cup,0]ss[0,Cap,0] =>0
[0,Cup,0]sf1_2fs[0,Cap,0] =>7
[0,Cup,0]f[0,Cap,0][0,Cup,0]f[0,Cap,0]
```

3 Movie Moves

Each of the Carter-Rieger-Saito movie-moves is associated with a positive integer which we shall call its **type**, which ranges from 1 to 31. For any subset U of $\{1, 2, ..., 31\}$, [LLT] defines the notion of two movies being (U)-regularly isotopic. We say M and M' are **$\mathcal{U}$-regularly isotopic** if it is possible to go from M to M' by a sequence of movie-moves NOT in $\mathcal{U}$. The C++ application programs we provide calculate $\mathcal{U}$-regularly isotopy invariants for the two cases, $\mathcal{U} = \{31\}$ and $\mathcal{U} = \emptyset$.

4 Input Movie Format

The application program that computes U-balanced isotopy invariants for movies expects the input movies to be described in an input file. The input file should contain a list of 1 or more movie descriptions.

The input is free format; that is, input lines can be split into several lines or combined and extra blanks or tabs are ignored. Newlines only affect comment lines as they terminate a comment.
Here is a grammar describing the format. $\{x\}$ means x can occur 0 or more times. $[x]$ means x is optional, and 'x' means the literal x should occur. An italicized *item description* represents an arbitrary value of the kind described by *item description*. Alternate values for a category are listed on separate lines.

MovieList ::= MovieDescription { MovieDescription }
MovieDescription ::= [MovieName] Movie `'#'`
MovieName ::= 'name' ':' *a movie name* ';'
Movie ::= Still { `'=>'` [*natural number*] Still }
Still ::= LblEvent { LblEvent }
LblFEvent ::= FEvent
's'
'f'
FEvent ::= '1_*natural number*'
'[' *natural number* ']'
'[' *natural number* ',' Event ',' *natural_number* ']'
Event := Cap
Cup
NE
NW

This grammar does not capture additional requirements for the 's' and 'f' labels.

1. The first still in a movie does not have any 'f' labels.
2. The last still in a movie does not have any 's' labels.
3. Any still in a movie other than the first or last has exactly 2 's' labels and 2 'f' labels.

Comments can be also inserted anywhere in the input file. Every thing from a % to the end of the line is ignored.

4.1 Example Input

Here is an example input file containing two movies.

The line numbers are **not** part of the file.

```
 1 % This file contains three movies.
 2 % second movie omits the optional movie name line.
 3 name:  a simple 2-unknot;
 4 % This comment line is ignored
 5 ss % This end of line comment is also ignored
 6 =>
 7 sf[0,Cup,0][0,Cap,0]sf
 8 =>
 9 ff
10 #
11 ss =>
12 f[0,Cup,0]ss[0,Cap,0]f =>
13 [0,Cup,0]fs[0,Cap,0][0,Cup,0]fs[0,Cap,0] =>
14 s[0,Cup,0]ff[0,Cap,0]s =>
15 ff
16 #
```

Lines 1, 2, and 4 are comment lines and line 5 contains a comment after initial movie fragment.

Line 3 is optional. The name provided here will be used to label the output. If the name line is omitted, as it is for the second movie, the label will simply be 'Movie n' for the n^{th} movie in the file.

Lines 10 and 16 each mark the end of a movie.

5 Programs

Included with the 2Knots C++ classes are three application programs:

- ProgramA
- ProgramB and programB0
- ProgramC and programC0

5.1 ProgramA

Our $\mathcal{U}$-balanced isotopy invariants are obtained by defining amplitudes of movies which depend on assignment of values from $\mathbb{Q}[q, q^{-1}]$to 102 *'normal coordinates'*. In order that such an assignment respect all the movie-moves not in $\mathcal{U}$, a set of linear equations determine by these movie-moves must be satisfied. ProgramA reads a file of movie-moves and generates this set of equations. A separate computer algebra system is then used to solve these equations.

5.2 ProgramB

ProgramB computes the $\mathcal{U}$-balanced isotopy invariants of each movie in an input file formatted as in Section 4.

Two choices of $\mathcal{U}$ are available, $\mathcal{U} = \{31\}$ and $\mathcal{U} = \emptyset$.

ProgramB computes the $\{31\}$-balanced isotopy invariants for any movie representing a 2-knot. There are 5 such invariants. ProgramB0 computes the $\emptyset$-balanced isotopy invariants for 2-knots.
There are are just 2 $\emptyset$-balanced isotopy invariants. Although these are invariant under all the movie-moves, it appears that they are uninteresting. The initial example movies we have computed suggest a conjecture that both the invariants computed in this case are just a multiple of the number of connected components of a 2-tangle. So in particular for a 2-knot, these invariants always appear to be equal to 1 as computed by programB0.

5.3 ProgramC

ProgramC accepts a pair of movie descriptions of 2-tangles and determines whether each of the 5 $\{31\}$-balanced isotopy invariants give the same value for the two 2-tangles. ProgramC0 accepts the same input as programC and performs the same computation, but for the 2 $\emptyset$-balanced isotopy invariants.

The input format for programC (and programC0) is the same as for programB except that exactly two movies should be input for programC.

6 Executing the Programs

After downloading the zip files containing the binary or the source versions of the 2Knots classes and programs. If installation has been done (see the installation instructions), the programs can be executed from a command prompt.

For example, to run programB you will need to have first prepared an input file of 1 or more movies formated as in Section 4. If the name of such an input file is **movies1.txt**, then you can run programB either like this:

```
programB movies1.txt

or simply

programB
Enter name of file containing a movie:

(In the secon case the program prompts for the input file.)
```

The output of the program goes to standard output and also to a file whose name is the same as the input file, but with '`.log`' appended. In the example above, the output be written to the file `movies1.txt.log`.

7 Examples

7.1 Sample input file for programB

Here is an example input file:

```
% This is file example.txt:
#
name:  Klein bottle;
ss =>
f[0,Cup,0]ss[0,Cap,0]f =>
[0,Cup,0]ssf[0,Cup,2];[1,Cap,1]f[0,Cap,0] =>
[0,Cup,0]f[1,Cup,1]s[0,Cap,2]f[0,Cup,2]s[1,Cap,1][0,Cap,0] =>
[0,Cup,0][1,Cup,1]sfsf[1,Cap,1][0,Cap,0] =>
[0,Cup,0]s[1,Cup,1]f[2,NE,0]s[2,NW,0]f[1,Cap,1][0,Cap,0] =>
[0,Cup,0]f[2,Cup,0][1,NW,1]fs[2,NW,0][1,Cap,1]s[0,Cap,0] =>
[0,Cup,0][2,Cup,0]s[1,NW,1]f[1,NE,1]s[2,Cap,0]f[0,Cap,0] =>
```

```
[0,Cup,0][2,Cup,0]sfsf[2,Cap,0][0,Cap,0] =>
[0,Cup,0]s[2,Cup,0]f[1,Cap,1]s[1,Cup,1]f[2,Cap,0][0,Cap,0] =>
[0,Cup,0]ffs[1,Cup,1][2,Cap,0]s[0,Cap,0] =>
s[0,Cup,0]ff[0,Cap,0]s =>
ff
#
name:  simple torus;
ss =>
f[0,cup,0]ss[0,cap,0]f =>
[0,cup,0]fs[0,cap,0][0,cup,0]fs[0,cap,0] =>
s[0,cup,0]ff[0,cap,0]s =>
ff
#
```

7.2 Sample execution - programB

Here is the sample execution of programB and the output. The output is sent to standard output and also to the output log file, **example.txt.log**. (In this sample, `$` is the command prompt.) The name of the file can alternatively be entered on the command line, in which case the program does not prompt for the file name.

```
$ programB
Enter name of file containing a movie: example.txt

------------------------------
Movie 1: Klein bottle

Invariant 0: 1*q^-4+ 2*q^0+ 1*q^4
Invariant 1: 1*q^-4+ 2*q^0+ 1*q^4
Invariant 2: 1*q^-4+ 2*q^0+ 1*q^4
Invariant 3: 1*q^0
Invariant 4: 1*q^0

------------------------------
Movie 2: simple torus

Invariant 0: 1*q^-4+ 2*q^0+ 1*q^4
Invariant 1: 1*q^-4+ 2*q^0+ 1*q^4
Invariant 2: 1*q^-4+ 2*q^0+ 1*q^4
```

```
Invariant 3: 1*q^0
Invariant 4: 1*q^0
```

7.3 Sample input file for programC

Input for programC should be in the same format as for programB, but should contain only 2 movies which will be compared. However, these movies are not restricted to representations of 2-nots and may represent more general 2-tangles. Here is a sample input file.

```
name: another movie move 1;
s[0,Cap,1]s =>
f[0,NW,1][0,Cap,1]f
#
s[0,NE,1][1,Cap,0]s =>
f[1,NW,0][0,Cap,1]f
#
```

7.4 Sample execution - programC

Here is the execution and output of running programC on the sample input file. Since the second movie did not have a 'name' line, it is simply reported as 'Movie2'. Both these movies have 3 input strings and so the amplitudes act on 3 dimensional 'bit vectors'. The two tangles in the sample file differ on the indicated bit vector.

```
$ programC tangle1.txt

------------------------------
Movie : another movie move 1
Movie : Movie2

Movies differ on input [ 0,0,1 ]  for invariant 0!

------------------------------
```