Management of Logical Functions with the Venn and VennCell Classes

by Paul Burchard
<burchard@math.utah.edu>

CLASS OVERVIEW

The Venn UI object provides a graphical way for users to select one of the 16 logical functions of two variables (such as AND, OR, XOR, etc). This is useful for management of query result sets and the construction of complex user selections.

The Venn object presents the user with a ``Venn diagram'' of two intersecting sets labeled with the names of the two variables. The highlighed regions of the diagram are the ones that evaluate to TRUE according to the logical function stored in the object. For example, the AND function would look like this:

paste_0.tiff ¬

The user changes the logical function by simply clicking or dragging the mouse across the regions whose states should be toggled.   The programmer can enter the most common logical functions through simple target-action connections in InterfaceBuilder, and general functions through methods described below.

The object is provided in two forms: as a Control and as an ActionCell.   The first version (Venn) is available on an Interface Builder palette for immediate usability and convenient localization.   The second (VennCell) is provided so that compound interface objects, such as Matrices of VennCells, can be created if necessary.   Both classes are accessed by identical methods, beyond those provided by the underlying Control and ActionCell classes.

PROGRAMMING METHODS

The logical function stored by the Venn object may be accessed as an integer code or directly as a Boolean operator (type `BOOLOP`):

```
typedef BOOL (*BOOLOP)(BOOL, BOOL, ...);
```

The methods which handle the Boolean operator form are:

```
- setStateFromOp:(BOOLOP)anOp;
- (BOOL)evalOp:(BOOL)arg1 :(BOOL)arg2;
- (int)argCount;
```

The `argCount` method always returns 2 for the classes described here.

The integer encoding of the logical function works like this: the *i*-th bit of the encoding is the value of the function when the arguments are set according to the bits of the number *i*. For example, the OR function is encoded by the number 14 = 1110_2 because it evaluates like this:

00	$\rightarrow$	0
01	$\rightarrow$	1
10	$\rightarrow$	1
11	$\rightarrow$	1

The methods which access the integer encoding are:

```
    - (int)state; // equivalent to intValue method
```

```
- setState:(int)value;
- takeStateFrom:sender; // copies sender's state
- takeStateFromIntValue:sender;
        // takes sender's intValue as its state
```

For Interface Builder convenience, the most common logical functions can be set using action methods.   The following are predefined:

```
- setStateReplace:sender; // y
- setStateRefine:sender; // x AND y
- setStateAdd:sender; // x OR y
- setStateRemove:sender; // x AND NOT y
- setStateReverse:sender; // NOT y
```

The designated initializers for the Control and ActionCell versions of the class are, respectively:

```
    - initFrame:(const NXRect *)frameRect; // Venn
    - initTextCell:(const char *)aString; // VennCell
```

(The argument aString should be the integer encoding of the logical function, converted into a text string.)

Finally, there are a variety of methods for setting and testing the appearance of the objects, such as the names of the two sets in the Venn diagram:

```
- setFont:fontObj;
- setBorderWidth:(float)width;
- (float)borderWidth;

// Names of the two sets
- takeFirstTitleFrom:sender;
- takeSecondTitleFrom:sender;
- setFirstTitle:(const char *)aString;
- setSecondTitle:(const char *)aString;
- (const char *)firstTitle;
- (const char *)secondTitle;
```

Dept. of Math, Univ. of Utah
Salt Lake City UT 84112 USA
801±581±6195
burchard@math.utah.edu (NeXTmail OK)