Adjusting slurs and ties in LilyPond

Improvements in \shape

Introduction

(read the introduction if you don't have experience with adjusting slurs in LilyPond)

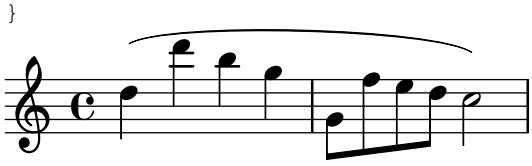
Slurs are particularly tricky elements of musical notation, because they are neither straight lines (like stems or beams), nor glyphs with fixed shape (like noteheads or clefs). Slurs – and ties, for that matter – are curves.

LilyPond, like most existing notation software, uses Bézier curves for typesetting slurs and ties. Such curves are defined by so-called "control-points"; usually four control-points are used for each curve. The first and last control-point define the beginning and end of the curve, while the middle points define the shape (curvature).

If you don't like how a slur (or tie) produced by LilyPond looks, you have to alter the control-points that define it. The most straightforward way is to explicitely specify the coordinates of each control-point manually, but this is very tedious (and if the layout of the score changes, the coordinates you had specified earlier will usually no longer make sense, so you will have to adjust them again).

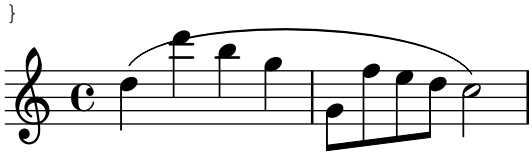
A much easier and more robust solution is to modify the default control-points' positions calculated by LilyPond. In 2012 David Nalesnik wrote a function called \shape for doing this. This function accepts a list of offsets that are applied to respective control-points. Let's look at an example showing how it's used – here's a case where default LilyPond slur looks ugly:

```
\relative c'' {
  d4( d' b g g,8 f' e d c2)
}
```



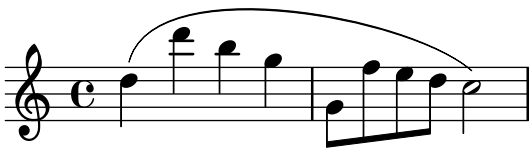
We begin by moving first and last control-points 2.3 staffspaces downward:

```
\relative c'' {
  \shape #'((0 . -2.3)(0 . 0)(0 . 0)(0 . -2.3)) Slur
  d4( d' b g g,8  f' e d c2)
}
```



The ends of the slur are now correct, but the middle is wrong. So, we move 2nd control-point 1 staffspace to the left and 3 staffspaces up, and the 3rd point 2 staffspaces to the left:

```
\shape #'((0 . -2.3)(-1 . 3)(-2 . 0)(0 . -2.3)) Slur
```

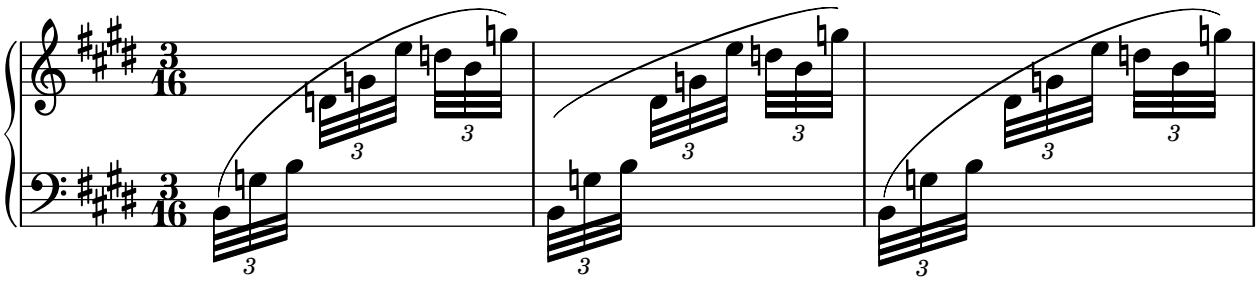


And we now have a correctly shaped slur. The function can also be used to adjust other curves – just specify appropriate object name (Slur, PhrasingSlur, Tie, etc.).

Making \shape even better

Real-life experience confirmed that \shape is an indispensable tool – I have already used it to create over a thousand publication-quality slurs. It was much, much less work than specifying control-points manually. Nevertheless, I came to the conclusion that the function could be improved to make it even more powerful and more robust against layout changes.

Take this example: first two measures are two nearly identical phrases (the difference is just one accidental) which get two drastically different slur shapes by default. 3rd measure contains the same phrase as the 2nd, but with changed spacing – again we get a different default slur (as of LilyPond 2.17.30):



None of these slurs has a satisfactory shape, especially the middle one. They could be adjusted using the method described above (by specifying how each control-point should be moved relative to its default position), but notice that to make all slurs similar, one would have to find three different sets of offsets (because each slur's default shape is different). What's worse, changes in score layout may have a “butterfly effect” on the slurs – they may change how LilyPond would typeset them by default, making user's offset values wrong. Such situation is much less likely when using offsets than in case of speficying control-points' coordinates explicitely, but it's not unheard of.

Improvements to \shape described below make it possible to adjust slurs more efficiently and in a more generic way. In particular, you'll see how to adjust all three slurs in the example above using just one \shape .

Summary of changes

There are three main changes in the way how \shape works.

Firstly, I've introduced some *shorthands* that simply reduce the amount of typing.

Secondly, there are now many different ways to specify the position of control-points:
- offsets relative to default positions (like in old \shape),
- symmetrical offsets,
- absolute coordinates,
- offsets relative to the noteheads,
- polar corrdinates.

Each control-point may be specified using a different method.

Since there are so many different ways of specifying control-points' positions, it doesn't make sense to call them “offsets”. In this document they will be called “instructions”. So, \shape accepts lists of instructions for each slur, with one instruction for each control-point: `\shape #'(instr-1 instr-2 instr-3 instr-4)` .

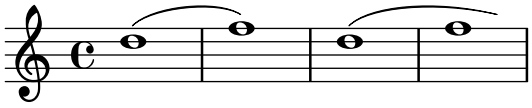
Finally, results of multiple \shape commands can now accumulate.

For compatibility's sake, the improved function is currently named \shapeII . I intend to add all these improvements to \shape function that is included in LilyPond, but before doing this I'd like them to be used for some time – maybe something will need adjustments. If I make any backward-incompatible changes, I will name the updated versions \shapeIII , \shapeIV etc. to avoid messing up users' scores.

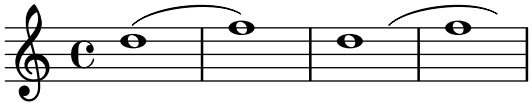
From now on, I'll use the following convention:
LEFT: default LilyPond curve **RIGHT:** result after applying \shape

Shorthands

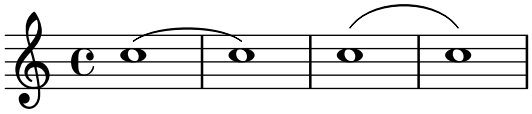
When you want to change just one control-point, typing `(0 . 0)` for all remaining points is very tedious. Now you can use `()` shorthand for every point that should remain unchanged. For example,
`\shapeII #'(()()()(3 . 0))` is equivalent to `\shapeII #'((0 . 0)(0 . 0)(0 . 0)(3 . 0))`:



When just one instruction is specified, it's applied to all points:
`\shapeII #'((3 . 0))` is equivalent to `\shapeII #'((3 . 0)(3 . 0)(3 . 0)(3 . 0))`:



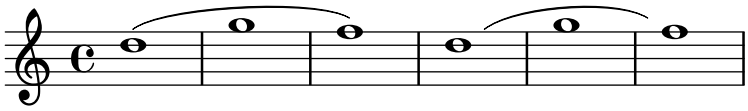
When just two instructions are specified, they are copied in reverse order:
`\shapeII #'(foo bar)` is equivalent to `\shapeII #'(foo bar bar foo)`. For example,
`\shapeII #'((0 . 1)(0 . 2))` produces:



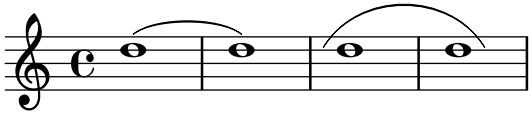
Symmetrical offsets

It often happens that you want to shape the curve symmetrically – for example, make it shorter. You could write it like this: `\shapeII #'((2 . 0)(2 . 0)(-2 . 0)(-2 . 0))` but it's a lot of typing, and it doesn't look elegant (the same number is repeated with different signs).

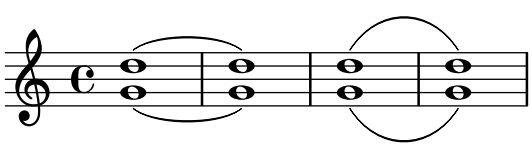
In such situations, use symmetrical offsets, which will be flipped horizontally for control-points on the right. Prefix numbers with with *symmetrical* (or just *s*), e.g.: `\shapeII #'((s 2 0)(s 2 0)(s 2 0)(s 2 0))` will shorten the slur by 2 staffspaces on each end. It can be written as `\shapeII #'((s 2 0))`:



`\shapeII #'((s -2 -1)(s -1 2))` is equivalent to `\shapeII #'((-2 . -1)(-1 . 2)(1 . 2)(2 . -1))`
:



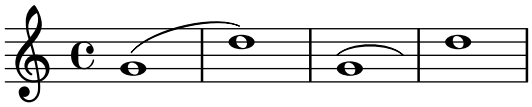
Also, offsets will be flipped vertically depending on curve direction. For upward curves
`\shapeII #'(()(s 0.5 2))` will be equivalent to
`\shapeII #'(()(0.5 . 2)(-0.5 . 2)())`, while for downward curves it will mean
`\shapeII #'(()(0.5 . -2)(-0.5 . -2)())`:



Absolute coordinates

You can explicitely specify the coordinates of a control-point, just as if you were directly overriding `control-points` property. To do this, start the instruction with the keyword *absolute*. You can also use *abs* or even *a* to save typing (all instructions demonstrated here have such shorthands defined).

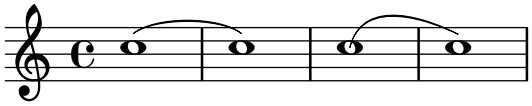
```
\shapeII #'((a 0 0)(a 1 1)(a 5 1)(a 6 0))
```



Relative to the noteheads

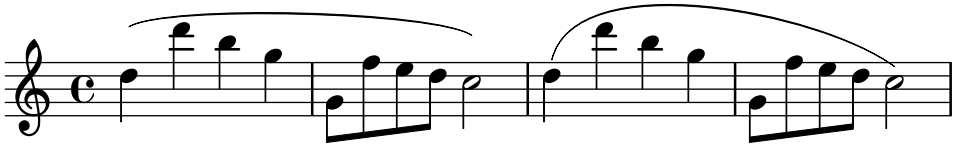
You can position control-points relative to respective noteheads. The offset is measured relative to the notehead center, and it is flipped like “symmetrical” offsets. First two points will use left notehead as the reference, and last two points will use right notehead:

```
\shapeII #'((head 0 0)(head 1 4)(h 2 2)(h 0 1))
```



You may omit the numbers and just write `(head)` . In that case, the point will be horizontally centered on the notehead, and vertically it will be about 0.7 staffspaces away from the notehead:

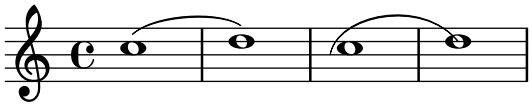
```
\shapeII #'((h)(-1 . 3)(-3 . 0)(h))
```



Mixing the styles

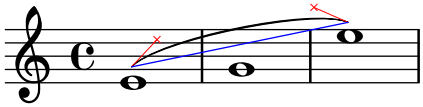
As you can see, different ways of specifying coordinates may be mixed:

```
\shapeII #'((a -0.5 0)(a 0 3)(s 1 1)(h 0 0))
```



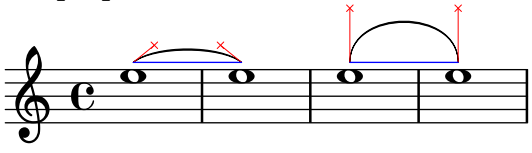
Polar coordinates

Arguably the most powerful way of specifying the positions of middle control-points is using polar coordinates. Use the following syntax: `(polar` *`angle radius`*`)` (Instead of “polar” you can use “p” to save typing). How angle and radius are measured? Look at the illustration below:



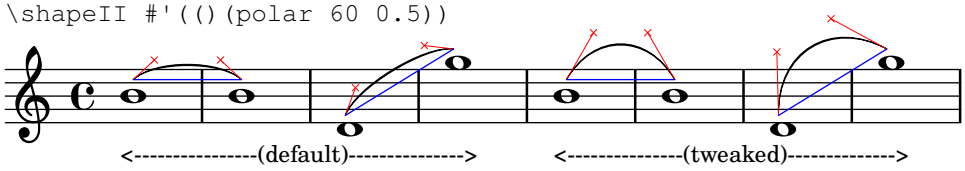
You can see a blue line connecting the ends of the slur. The length of this line is what we call *slur length* , and the angle between this line and the horizontal direction is *slur slope* . In polar coordinates, radius is measured in slur length; angle is measured in degrees, relative to the slur slope.

In the following example, the distance between 1st and 2nd control-points (as well as between 3rd and 4th) is 0.5 of the blue line's length, and the red lines (which connect 1st point to the 2nd, and 3rd point to the 4th) are perpendicular to the blue line: `\shapeII #'()(polar 90 0.5)(polar 90 0.5)())`



Here's another example, showing how the situation changes depending on the slur slope – notice the 60 degrees angle between the blue and red lines:

```
\shapeII #'(()(polar 60 0.5))
```

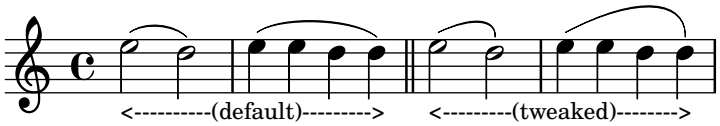


It is possible to measure the angle against the horizontal line by using “absolute” polar coordinates:

```
\shapeII #'((0 -2)(absolute-polar 88 0.5)(ap 20 0.2)())
```

With polar coords, the same values can be used for different slurs to produce very similar shapes:

```
\shapeII #'(()(p 30 0.7)(p 90 0.3)())
```

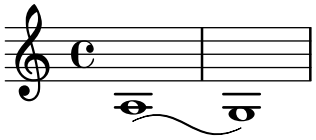


You can specify a point's polar coordinates relative to its default polar coordinates. For example, this command will place 2nd point two times farther and 20 degrees more outwards than default placement:

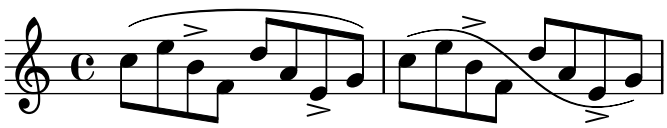
```
\shapeII #'(()(rp 20 2)()()) Slur
```

S-shaped slurs are easy to achieve with polar coordinates:

```
\shapeII #'(()(p -30 0.5)(p 30 0.5)())
```



```
\shapeII #'((h 0 1.8)(p 35 0.5)(p -35 0.5)(h 0 -1.5))
```



As you can see, shorthands work with polar coordinates:

```
\shapeII #'(()(p 50 0.35))
```

Accumulation

\shape can be written in two forms – *override-like* :

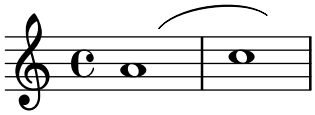
```
{
  c''1 ( c'')
  \shapeII #'((0 . 2)) Slur
  c''1 ( c'')
}
```

and *tweak-like* :

```
{
  c''1 ( c'')
  c''1-\shapeII #'((2 . 0)) ( c'')
}
```

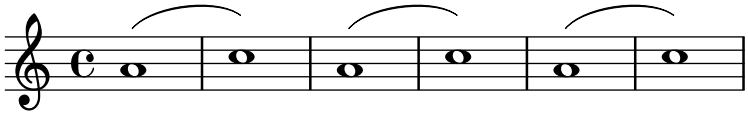
If you use both *tweak-like* and *override-like* \shape on a slur, their effects will combine:

```
{
  \shapeII #'((0 . 2)) Slur
  a'1-\shapeII #'((2 . 0)) ( c'')
}
```



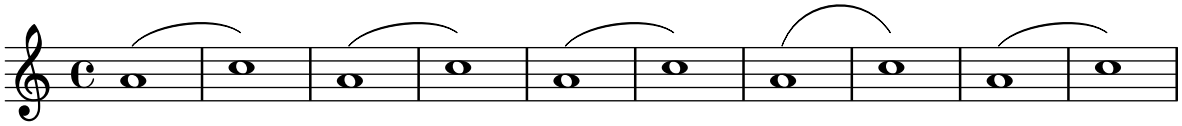
Since LilyPond 2.17.29, the *override-like* form will affect *all* subsequent curves of the specified type (see Issue 3603). For example, this will move all three slurs up:

```
{
  \shapeII #'((0 . 2)) Slur
  a'1 ( c'')
  a'1 ( c'')
  a'1 ( c'')
}
```



This gives us interesting possibilites. If we have a passage of similar slurs, we can use the override-like \shape to give all the slurs some particular appearance, and then use tweak-like \shape to fine-tune any slurs for which the “general” \shape didn't provide satisfactory results. In the example below, first all slurs are moved upwards, and then one of them is given more curvature:

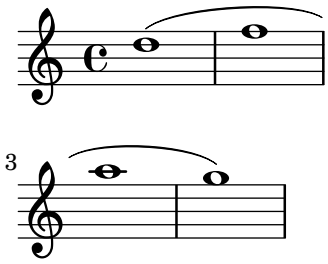
```
{
  \shapeII #'((0 . 1.5)) Slur
  a'1 ( c'')
  a'1 ( c'')
  a'1 ( c'')
  a'1-\shapeII #'(()(rp 20 1.7)) ( c'')
  a'1 ( c'')
}
```



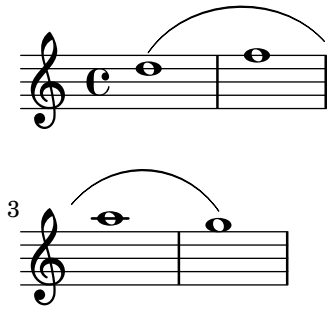
Broken curves

All of this should work for broken slurs/ties as well.
If you specify one set of instructions, it will be applied to all siblings:

default:



```
\shapeII #'(()(rp 15 2))
```



Epilogue

Equipped with all these features, we can easily fix the bad-looking slurs from the example at the beginning. Note that just **one** override gives **all** slurs correct appearance:

```
\shapeII #'((h)(p 55 0.5)(p 50 0.2)(h 0 1.5)) Slur
```