File: **override-object-property.ly**

Understanding the difference between **\override** and **\overrideProperty** is famously difficult. But it seems like the distinction may be at least a little easier to understand than is usually thought. I'm not completely certain about what I'm concluding here, but let's see.

It seems that the confusions may come from a subtle misunderstanding in the way that users think of the **\override** command. The result of **\override NoteHead.color = #red** is red note heads. But how does this result come about? The **\override** command sets a default value *that will later be used when objects are created* , but the **\override** command *does not (directly) modify the property of any (already-existent) object.* ; rather, the **\override** command sets a default value (stored at the level of the voice, staff, score or other context) *that will be consulted when objects are created at some later time in the future.* In other words, **\override** doesn't operate on in-memory objects (like note heads); **\override** operates on the bank of default settings that the object-creating part of LilyPond consults when it is time to create new in-memory objects.

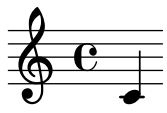
This means that, at the moment that **\override NoteHead.color = #red** is read, the command does absolutely nothing to any in-memory note head objects that LilyPond has already created; rather, the command changes the default settings that will be used when LilyPond creates more note heads in the future.

The problem is that there is normally no motivation for users to make the distinction that we're drawing here: users can -- and almost certainly *do* -- think something like " **\override NoteHead.color = #red** produces red note heads"; there's absolutely no reason for users to keep in mind (or even know in the first place) that overrides override defaults rather than the properties of in-memory objects themselves.

But this distinction -- overriding the defaults used to create objects versus overriding the properties of objects once they have been created -- is perhaps the only way to understand why LilyPond has an **\overrideProperty** command: in 99 cases out of 100, users will **\override** object-creation defaults; but in a small number of cases it will be possible (or even required) to use **\overrideProperty** to change the properties of in-memory objects that have already been created.

Jean's guide to extending LilyPond includes this example ...

```
{
  \override NoteHead.staff-position = -10
  c'4
}
```



... which does nothing, and which interprets without raising an error. Given that note heads have a staff position, *why doesn't the override do anything?* And given that the override doesn't do anything, *why does the example compile without warning?*

It's in cases like these that they distinction between **\override** and **\overrideProperty** becomes clear.

`\override NoteHead.staff-position = -10` does nothing because **`\override`** is used to set default values for object creation, and because LilyPond evidently does not model note heads as having a default staff position. This is, to be clear, a modeling decision (one could imagine note heads taking a default staff position of, say, 0), but it's a modeling decision that makes sense because the staff position of a note head almost always results from the note head's pitch in combination with the clef on which the note is spelled. LilyPond certainly models note heads as having a staff position. But the relevant point here is that LilyPond only sets that staff position of a note head ***after*** the note head has been created as an object in memory.

And what about once the note head has been created as object in memory? This is precisely the "moment" (or "time") during LilyPond's execution at which it becomes possible to modify the staff position of a note head. This is precisely what **`\overrideProperty`** is used for:

```
{
  \overrideProperty NoteHead.staff-position -10
  c'4
}
```

The example isn't well-motivated (why change the staff position of a note head instead of its pitch?), but it shows what **`\overrideProperty`** can do that **`\override`** can not.

In summary, thinking of **`\override`** as **`\overrideDefault`** might help explain what the function actually does, and why **`\overrideProperty`** is able to change the properties of objects after they've been created. All of this comes from thinking about **`\override`** as operating on objects to be created in the future and **`as`** operating on objects that have already been created.

But even with all of this, the distinction between the functions is almost certain to remain obscure to almost all users. Why? Because thinking about whether LilyPond has already created an object or whether LilyPond is soon to create an object involves creating a mental model of the (invisible) steps LilyPond is going through as it works, rather than just thinking about the appearance of objects on the page.