express the similarity between the pattern A and the pattern B by expressing the pattern B as a copy of the pattern A positioned differently within our space.

We now would like to express the fact that the rhythm in the first bar A1 is the same as in the bar A2. To do so, we notice that the bar A2 looks like a projection on the axis of pitches of the bar A1, as illustrated at figure 8

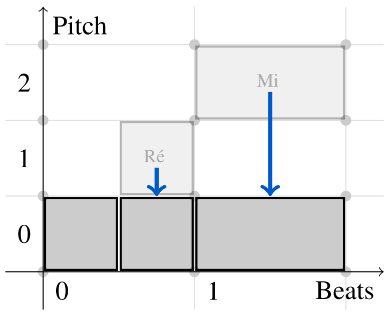

Figure 8. Projection of the bar A1 on the axis

A first idea, which turns out to be wrong, would be to see the projection as a coordinate change and write the following code:

```
  wrongA = do
    tileA1;
    change project;
    tileA1;
```

This code could indeed yield the pattern A. However, after applying project the coordinate system has lost one dimension, and as a consequence all the following notes will be E. To solve this problem, we need to specify which notes we want to apply the projection to.

5.1 Applying transformations

To be able to use irreversible transformations like the projection, we first group the elements we want to transform, and then apply the transformation.

Transformations are also a polymorphic parameter. They are provided in a similar way we specified CoordChanges. In our code, we provide a type Transforms that enables us to use project and inverse in addition to coordinate changes.

For instance, in order to express the rhythmic similarity between A1 and A2, we can express the pattern A in a hierarchical way:

```
type Music = Media Space CoordChanges Transforms
     Duration

hierarchyA :: Music
hierarchyA = do
    group idle tileA1;    -- A1
    change (delay 2);
    change (transpose 2);
    group project tileA1  -- A2
```

Let us emphasize that the `group` instruction forgets about the coordinate change inside the group, which explains why we had to use relative positioning.

In the previous code, we first group together the elements of A1. The idle transformation does not do anything, so `group idle tileA1` groups tileA1 into a single object, and forget about the coordinate changes that happened in tileA1.

Since we forgot about the coordinate change in tileA1, we need to use relative positioning again to properly position the projection of A1, which gives us the pattern A2.

5.2 Complete encoding of *Do ré mi, la perdrix*

In order to get a complete hierarchical encoding of *Do ré mi, la perdrix*, we need to introduce the *inversion* in order to express the similarity between pattern A and pattern C. The inversion corresponds to a vertical symmetry as represented at figure 9. The patterns C and D can be obtained by applying an inversion to the pattern A, then use translations in order to position them properly.

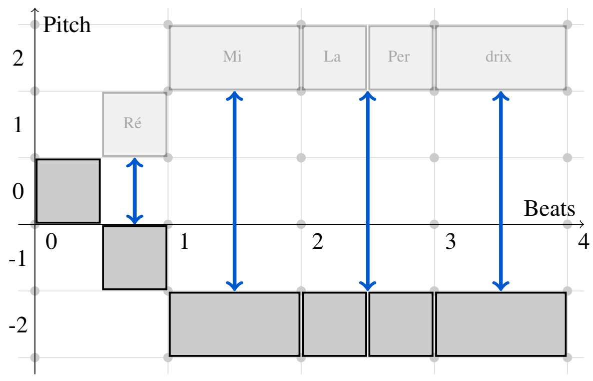

Figure 9. The inversion shows the similarity between the pattern A and the pattern C

We are now ready to completely encode the structure of *Do ré mi, la perdrix* proposed at the very begining of this article in figure 1.

```
fullSong :: Media Space CoordChanges Transforms
    Duration
fullSong = do
      idle |-> tileA;     -- A
      idle |-> tileA;     -- B

      change (transpose (-1));
      inverse |-> tileA   -- C

      change (transpose 1);
      inverse |-> tileA   -- D
```

Since hierarchyA was built only using A1, we expressed the whole song only by transforming copies of the A1 pattern ! As a consequence, we succeded in encoding both the structure and the similarity relations between the elements of the structure.

6. SEMANTICS

In all previous sections, we have tried to give a concrete intuition of what the semantics of the modeling language presented here should be. In all our examples, one may have observed that our pieces of music are defined by *programs*, themselves defined as lists of *elementary instructions* separated by a semicolon. We provide here some more details on what our polymorphic implementation actually is, together with elements underlying its semantics. The generic types we use, their relationship and related properties are reviewed throughout this section.