# ABC with a UNIX flavor

Bruno M. Azevedo
J. João Almeida

Departament of Informatics,
Universidade do Minho,
Portugal

June 21, 2013

## Structure of this presentation

## Introduction

- ABC

  `Textual music notation`

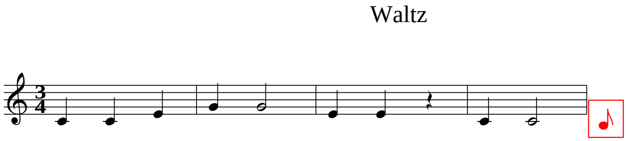
- UNIX metaphor

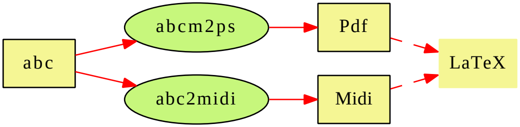
| | |
|---|---|
| `Simple single-task programs` | `Simple music single-task programs` |
| `filters` | `filters` |
| `cat, paste, grep, wc` | `abc-cat, abc-paste, abc-grep` |
| `Compositionality, pipes` | |
| `Universal type: text` | `Universal music (abc)` |
| `Development Language (C)` | `Devel DSL  Abc::DT` |

- ABC::DT:
  - Rule-based DSL
  - ABC oriented
  - Compact specification
  - Deals with real ABC music
  - Surgical processing
  - Rich embedding mechanisms (Perl)

# Abc music notation

```
X:1
M: 3/4
T: waltz
K: C
C C E | G G2 | E E z | C C2 |
```





## Related Projects and Tools

- abcm2ps

  `Translates ABC music into sheet music scores in PostScript or SVG`

- abc2midi

  `Converts ABC music into a MIDI file`

- Music21

  `A Python-based toolkit for computer-aided musicology`

- Haskore

  `Haskell modules for creating, analyzing and manipulating music`

## Internal Music Representation

- Partwise (melody)
- Timewise (harmony)
- Sourcewise
- monads

## ... Internal Music Representation

**Partwise Structure: organized by the part**

score $\rightarrow$ part*

part $\rightarrow$ (meta, note*)

note $\rightarrow$ (freq, duration,...)

**Timewise Structure**

score $\rightarrow$ harmonic-instant*

harmonic-instant $\rightarrow$ (duration, (part$\rightarrow$ note), ...)

- The score is organized by a vertical element;

**Sourcewise Structure: follows the ABC source order**

score $\rightarrow$ (abc-element, position)*

abc-element $\rightarrow$ meta | note | ...

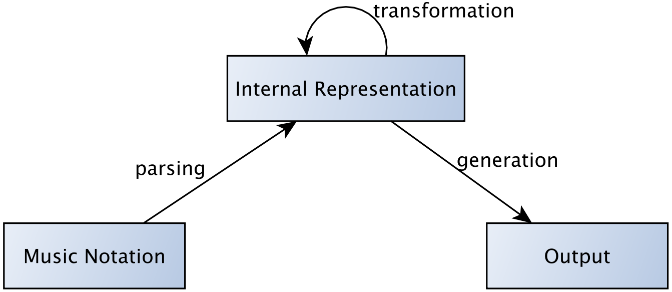
position $\rightarrow$ (state, current-time, current-part, ...)

- Enables the easy reconstruction of the original ABC;
- Needs calculation of current position (time, part, etc)

## ABC processing tool based in ABC::DT

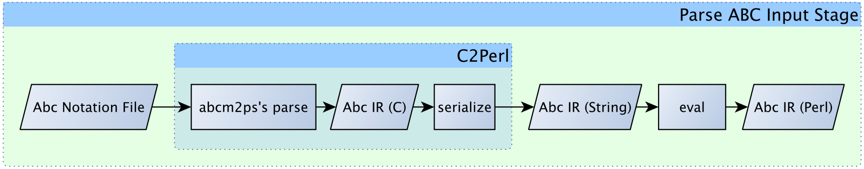
An ABC processing tool follows a traditional compiler's structure:

- Parse ABC input
- Transform the generated representation (based on ABC::DT rules)
- Generate the output



## Parse ABC Input

`abcm2ps`' parser is the parser currently being used by an ABC processing tool. Its generated IR is a sequential structure, in which each element corresponds to an ABC element.
That structure - a list of C data structures - has to be adapted to Perl.



## Transform the generated representation

- Traversal of the IR applying ABC::DT rules to each abc element.
- elements not covered by the actuators are kept unchanged.
- Effective in building tools that do simple transformations:
    - just provide the points and the specific changes.

## ABC::DT Rules

**ABC::DT Rule**

actuator ⇒ transformation

- `actuator` selects an ABC element or a group of elements.
- `transformation` specifies how each element should be processed

```
note
in_line::K:
note::!f!
V:Tenor::note
V:Tenor::note::_B
```

### Special Actuators

- -default Describes how to transform each ABC element which is not covered in the traversal
- -end Enables a general post processing of the final ABC, hence, making possible to attain different output formats

# Processor Algorithm

```
abc-dt-processor(abc-file,handler)
Input: abc-file
Input: handler: [($actuator, transform$)]                    //rules
$musicIR \leftarrow$ parse(abc-file)                         //[abc-element]
forall the $a \in musicIR$ do
    $state \leftarrow$ recalculate current position
    $r \leftarrow$ rule $\in$ handler with best matching actuator or -default
    $a \leftarrow r.transform(a, state)$
return handler[-end]( $abc(musicIR)$)
```

During the structure's traversal,

- when an element matches an actuator, the corresponding transformation is applied.
- the current element's state is calculated.

## Generate the output

The transformed IR is outputted.
The identity function - `toabc()` - is based on `tclabc`'s implementation.

## ABC::DT by example

### All But One

- Reduce the volume of my voice
- To help musicians in individual rehearsal of multi-voice music.
- Add a midi directive

**All But One: ABC::DT rules**

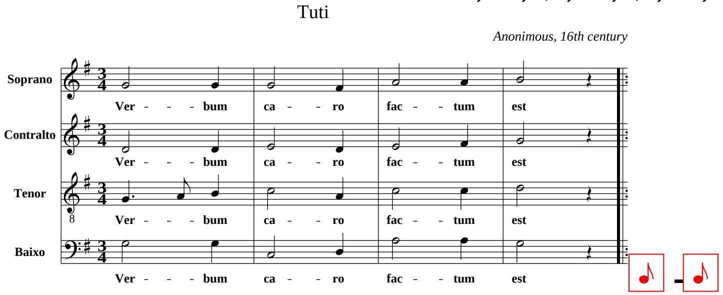
```
  "V:Tenor" => sub{ toabc() . "%%MIDI control 7 0\n"; }
```

```
...
my %handler = (
  "V:$voice" => sub{ toabc() . "%%MIDI control 7 $min_volume\n"; }
);
abcdt($file,%handler)
...
```

```
T:Tuti
C:Anonimous, 16th century
M:3/4
K:G
V:1 name="Soprano" clef=treble
G4 G2|G4 F2|A4 A2|B4 z2|:
w: Ver- bum|ca- ro|fac- tum|est|
V:2 name="Contralto" clef=treble
D4 D2|E4 D2|E4 F2|G4 z2|:
V:3 name="Tenor" clef=treble-8
G3 A B2|c4 A2|c4 c2|d4 z2|:
V:4 name="Baixo" clef=bass
G,4 G,2|C,4 D,2|A,4 A,2|G,4 z2|:
```

$\rightarrow$

```
T:Tuti
C:Anonimous, 16th century
M:3/4
K:G
V:1 name="Soprano" clef=treble
G4 G2|G4 F2|A4 A2|B4 z2|:
w: Ver- bum|ca- ro|fac- tum|est|
V:2 name="Contralto" clef=treble
D4 D2|E4 D2|E4 F2|G4 z2|:
V:3 name="Tenor" clef=treble-8
%%MIDI control 7 25
G3 A B2|c4 A2|c4 c2|d4 z2|:
V:4 name="Baixo" clef=bass
G,4 G,2|C,4 D,2|A,4 A,2|G,4 z2|:
```



## ABC-paste

### ABC Paste

Merges the voices of tunes parallel to each other in the time perspective.

**Algorithm**

1. get the header for the resulting score
   ```
   First tune with at least one note written;
   ```
2. Paste the tunes
   ```
     remove redundant metadata
     calculate part length
   ```
3. Append any necessary rests
   ```
     Append a multi-measure rest if voice is shorter than the longest
   ```

## ABC-cat

### ABC Cat

Concatenates each tune one after the other in the time perspective.

### Algorithm

1. get the header for the resulting score

```
First tune with at least one note written;
```

2. For each tune, print the tune and aditional rests

```
Keeps track of each voice's id, measures, meter (M),
  length (L), key (K) and tempo (Q);
Appends rests to any voice that isn't present in the
  current tune;
Appends rests to any voice belonging to the current tune
  that is not present in previous tunes;
```

## Composition of ABC Paste and Cat

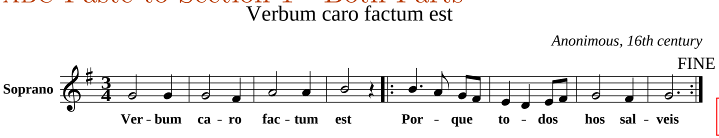
Assemble the whole score by composing ABC Paste with ABC Cat.

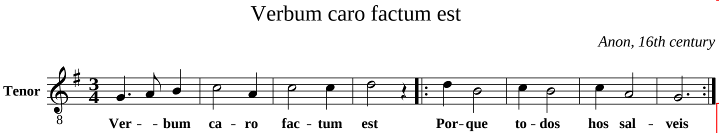
**Score**

`Verbum caro factum est`

- assemble three sections of the score (1, 2 and 3)
- for two parts only (Soprano and Tenor).
- Each part and section is written in single files.

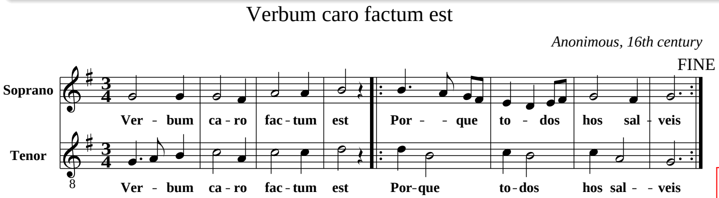
## ABC Paste to Section 1 - Both Parts


Verbum caro factum est
Anonimous, 16th century
FINE
Soprano
Ver - bum ca - ro fac - tum est Por - que to - dos hos sal - veis

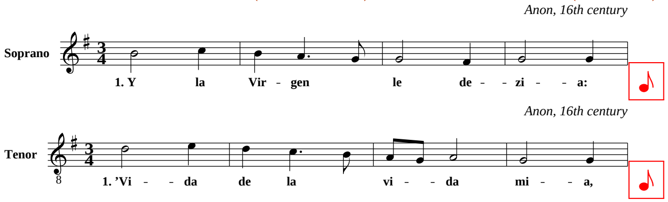

Verbum caro factum est
Anon, 16th century
Tenor
Ver - - bum ca - ro fac - tum est Por - que to - dos hos sal - veis

### ABC Paste

```
abc_paste 101.abc 103.abc > A.abc
```

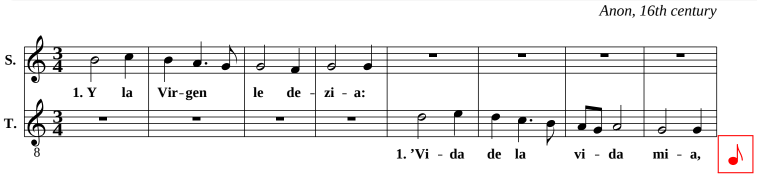

Verbum caro factum est
Anonimous, 16th century
FINE
Soprano
Ver - bum ca - ro fac - tum est Por - - que to - dos hos sal - veis
Tenor
Ver - bum ca - ro fac - tum est Por - que to - dos hos sal - - veis

## ABC Cat Section 2 (Soprano) and Section 3 (Tenor)


Anon, 16th century
Soprano
1. Y la Vir - gen le de - - zi - - a:
Anon, 16th century
Tenor
1. 'Vi - - da de la vi - - da mi - - a,

### ABC Cat

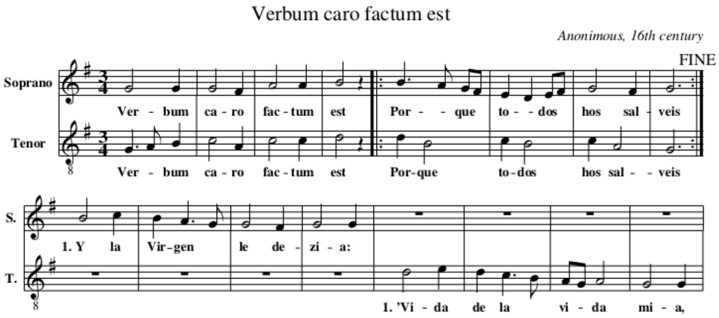
```
abc_cat 201.abc 303.abc > b.abc
```


Anon, 16th century
S.
1. Y la Vir-gen le de - zi - a:
T.
1. 'Vi - da de la vi - da mi - a,

## Applying ABC Cat to previous outputs

**ABC Cat**

```
abc_cat a.abc b.abc > c.abc
```


Verbum caro factum est
Anonimous, 16th century
FINE
Soprano
Tenor
Ver - bum ca - ro fac - tum est Por - - que to - - dos hos sal - veis
Ver - bum ca - ro fac - tum est Por-que to-dos hos sal - - veis
S.
T.
1. Y la Vir-gen le de - zi - a:
1. 'Vi - da de la vi - da mi - a,

A

B

## Conclusions

- Reusing `abcm2ps`'s parser was very important to help guarantee this work's quality, coverage and developing time.
- Using Perl as the language embedded into ABC::DT provides a rich environment to allow easier processing of text. Furthermore, through the use of data structures, like hashes, the user has bigger expressive power to specify transformations.
- We believe that the rule based processor makes it possible to write very compact tools.
- The existence of DSL's like ABC::DT helps to the simplification of crafting new ABC processing tools.

## The End