Key Detection Algorithm

Tongyu Lu, 26/Jun/2022

Main Task and Methodology

Suppose the chord symbol detection system has been developed. We observe a chord progression $\mathbf{c} = [(r_i, T_i, b_i), i = 0,1, \dots, |\mathbf{c}| - 1]$, where r_i stands for the root note, T_i stands for the chord type string, and b_i stands for the bass note. Next, given this chord progression, how to determine the key context of each chord?

This task in general is ill-defined, because many cases can have multiple explanations, and can even have no tonal minor/major key based explanation. For example, given a chord progression $C - G - C - G$, can you surely say that it is in $\mathrm{C-Major}$ key? Not really. It can also in $\mathrm{G-Major}$ key, with functions $S - T - S - T$. However, $\mathrm{C-Major}$ is more probable, because it is the default explanation for most pieces.
Therefore, a key-detection system should be able to give the most probable key determination if possible; otherwise, it should give "unknown key".

To formulate the problem, we directly introduce the algorithm we use, and the explain every element in this algorithm.

Algorithm: rule-based key detection given symbolic chord progression

```
Input: a chord progression $\mathbf{c} = [(r_i, T_i, b_i), i = 0,1, \dots, |\mathbf{c}| - 1]$
Output: a function-key explanation $\boldsymbol{\kappa} = [(\zeta_i, \kappa_i), i = 0,1, \dots, |\mathbf{c}| - 1]$
1.  For each binary chord progression $(r_i, T_i, b_i) \to (r_{i+1}, T_{i+1}, b_{i+1}), i = 0,1, \dots, |\mathbf{c}| - 2$, get all possible guesses of chord
    functions, $g_i = [(\zeta_{i_0}, \kappa_{i_0}), \dots]$, and assemble all such guesses into function-key-tuples-list $\mathbf{g} = [g_0, \dots g_{|\mathbf{c}|-1}]$

2.  For each (non-repeating) quaternary chord progression at indices $[i, i+\Delta_1, i+\Delta_2, i+\Delta_3], i = 0,1, \dots, |\mathbf{c}| - 4$:
3.      Check the corresponding function-key-tuples in $\mathbf{g}$
4.      If a form $(\zeta_{i_*}, \kappa), (\zeta_{(i+\Delta_1)_*}, \kappa), (\zeta_{(i+\Delta_2)_*}, \kappa), (\zeta_{(i+\Delta_3)_*}, \kappa)$ is detected
        and $\zeta_{i_*}, \zeta_{(i+\Delta_1)_*}, \zeta_{(i+\Delta_2)_*}, \zeta_{(i+\Delta_3)_*}$ is key-determining:
5.          Rewrite $g_j$ with $[(\zeta_{j_*}, \kappa)]$, $j = i, \dots, i+3$

6.  For each (non-repeating) ternary chord progression at indices $[i, i+\Delta_1, i+\Delta_2], i = 0,1, \dots, |\mathbf{c}| - 3$:
7.      visit_nearby_deterministic_key$(i+2)$
8.      Check the corresponding function-key-tuples in $\mathbf{g}$
9.      If a form $(\zeta_{i_*}, \kappa), (\zeta_{(i+\Delta_1)_*}, \kappa), (\zeta_{(i+\Delta_2)_*}, \kappa)$ is detected and $\zeta_{i_*}, \zeta_{(i+\Delta_1)_*}, \zeta_{(i+\Delta_2)_*}$ is key-determining:
10.         Rewrite $g_j$ with $[(\zeta_{j_*}, \kappa)]$, $j = i, \dots, i+2$

11. mark_reapeating_chords$(\mathbf{g})$
12. align_chord_with_detected_keys$(\mathbf{g})$
13. return prune_function_key_tuples_list$(\mathbf{g})$
```

To address a few immediate confusions:

- ζ stands for the binary chord function tag. For example, given chord progression $D^{-7} - G^7 - C^7_\Delta$, it has binary chord function tags $[\text{S-D, D-T, -}]$

- The basic idea of this algorithm: first detect deterministic chord progressions with functions as “bases”; then start from those “bases” and use the inertia of key perception, convert the nearby keys into the keys of bases; finally, remove duplicate chord progressions and return the final results.

If still confused, it is OK. In the next sections, the building-ups will be introduced, and the mechanism of this algorithm will be demystified.

Chord Functions

In tonal music, each chord has its function. Given a key, e.g., $\mathrm{C-Major}$, then the chord progression $C-D^{-}-F-G-C$ has function $\mathrm{T-S-S-D-T}$, standing for “tonic”-“subdominant”-“subdominant”-“dominant”-“tonic”.
In Major keys, different chord roots and different chord qualities can have different functions. For example, in $\mathrm{C-Major}$ key, we have a list of basic functions:

- C Major chord family has tonic function (T)
- $D\mathrm{b}$ Major chord has “Napoleon” function (N), and $D\mathrm{b}$ dominant or altered chords have dominant function
- D minor chord family has subdominant function (S)
- D dominant chord family has “double-dominant” function (DD)
- E minor chord family has median function
- F Major chord family has subdominant function
- G Major or dominant family have dominant function
- A minor family have subdominant function
- B dim family has dominant function
- All leading seventh chords of G dominant have dominant function
- All leading seventh chords of D dominant have double-dominant function

However, it is not quite easy to realize “given chord progression without key, output the functions”. How to do this? We need to guess. For example, observing $G-C$, we can guess that it could be $(\mathrm{S\text{-}D}, \mathrm{F-Major})$ or $(\mathrm{D\text{-}T}, \mathrm{C-Major})$ or $(\mathrm{T\text{-}S}, \mathrm{G-Major})$. With this single progression, we are not able to determine either of them. As a result, we write them up in an assemble $[(\mathrm{S\text{-}D}, \mathrm{F-Major}), (\mathrm{D\text{-}T}, \mathrm{C-Major}), (\mathrm{T\text{-}S}, \mathrm{G-Major})]$, and proceed on other binary chord progressions.

To implement this procedure, such binary chord progression guesses are encoded as such:

1. Define rules of determining single-chord possible functions for both major and minor keys. For example, Tonics in Major keys can only be Major-family chords; Dominants in Major keys can be Major triads or Dominant-family chords. For instance, function $\mathrm{can\_be\_major\_subdominant\_IV(chord\_symb)}$ checks whether $\mathrm{chord\_symb}$ is in Major family.
2. Define rules of guessing binary functions. First define rules like
$$\mathrm{can\_be\_V\_I\_major(this\_root, root\_step, this\_symb, next\_symb)}$$
which means: given the step from this chord root to next chord root, first check whether it is the modulo of downward fifth interval; if so, check whether this chord “$\mathrm{can\ be\ major\ dominant\ V}$” and whether the next chord “$\mathrm{can\ be\ major\ tonic\ I}$”; if so, return function-key pair $(\mathrm{D\text{-}T}, (\mathrm{this\_root} + \mathrm{root\_step}) - \mathrm{Major})$; if not, return False.
    - For each possible root step, define the binary checking rules in format as introduced; only consider binary chord progressions which are commonly used (e.g., $\mathrm{S\text{-}D}$ progression is not defined).
3. Then, given a binary chord progression, search among the defined rules for fitted root step, and assemble the results of the fitted rules.

This procedure is the first step in the key determining algorithm. Get back to check it!
If interested, check the code for details.

Key-determining Chord Progressions

After having binary possible guesses, we proceed on determining key as bases.
We start from a simple example $C - C/E - F - G - C$. After the binary function guess procedure, we have a resulting "function-key-tuples-list":

$$\mathbf{g} = \begin{bmatrix} [(\text{STAY}, \text{UNK\_KEY})], \\ [(\text{D-T}, \text{F}), (\text{T-S}, \text{C}), (\text{DD-D}, \text{Bb}), (\text{DD-D}, \text{a\#})], \\ [(\text{S-D}, \text{C}), (\text{T-DD}, \text{F})], \\ [(\text{D-T}, \text{C}), (\text{T-S}, \text{G}), (\text{DD-D}, \text{F}), (\text{DD-D}, \text{f})], \\ [(-, \text{UNK\_KEY})] \end{bmatrix}$$

Next, we go through $\mathbf{g}$, and see if we can detect any key-determining chord progressions.
The term "key-determining chord progression" stands for a set of special chord progressions that are enough to detect a key. For example, given D^{-7}, G^7, C^7_Δ, a trained musician will immediately say that it is a $\text{II} - \text{V} - \text{I}$ progression in $\text{C} - \text{Major}$.
Instead of encoding the chord symbols as chord progressions, we encode the chord functions as chord progressions. For example, the $\text{II} - \text{V} - \text{I}$ progression can be translated into $\text{S} - \text{D} - \text{T}$ progression. In other words, whenever we encounter a $\text{S} - \text{D} - \text{T}$ progression, we know that it is in the key of the T chord root.
We start from quaternary key-determining chord progressions, including

$$\begin{bmatrix} (\text{T-S, S-D, D-T}), \\ (\text{STAY, S-D, D-T}), \\ (\text{T-S, S-DD, DD-T}), \\ (\text{T-DD, DD-D, D-T}), \\ (\text{T-D, D-T, T-D}), \\ (\text{T-S, S-T, T-S}) \end{bmatrix}$$

Then, we check whether $\mathbf{g}$ contains any quaternary key-determining chord progression, where all binary progressions within the quaternary progression share the same key. Here, we find one:

$$\begin{bmatrix} [(\text{STAY}, \text{UNK\_KEY})], \\ [(\text{D-T}, \text{F}), \color{red}{(\text{T-S}, \text{C})}, (\text{DD-D}, \text{Bb}), (\text{DD-D}, \text{a\#})], \\ [\color{red}{(\text{S-D}, \text{C})}, (\text{T-DD}, \text{F})], \\ [\color{red}{(\text{D-T}, \text{C})}, (\text{T-S}, \text{G}), (\text{DD-D}, \text{F}), (\text{DD-D}, \text{f})], \\ [(-, \text{UNK\_KEY})] \end{bmatrix} - \begin{bmatrix} \color{red}{(\text{T-S, S-D, D-T})}, \\ (\text{STAY, S-D, D-T}), \\ (\text{T-S, S-DD, DD-T}), \\ (\text{T-DD, DD-D, D-T}), \\ (\text{T-D, D-T, T-D}), \\ (\text{T-S, S-T, T-S}) \end{bmatrix}$$

If readers are careful enough, they may find that there is also a plausible one

$$\begin{bmatrix} [(\text{STAY}, \text{UNK\_KEY})], \\ [\color{blue}{(\text{D-T}, \text{F})}, (\text{T-S}, \text{C}), (\text{DD-D}, \text{Bb}), (\text{DD-D}, \text{a\#})], \\ [(\text{S-D}, \text{C}), \color{blue}{(\text{T-DD}, \text{F})}], \\ [(\text{D-T}, \text{C}), (\text{T-S}, \text{G}), \color{blue}{(\text{DD-D}, \text{F})}, (\text{DD-D}, \text{f})], \\ [(-, \text{UNK\_KEY})] \end{bmatrix} - \begin{bmatrix} (\text{T-S, S-D, D-T}), \\ (\text{STAY, S-D, D-T}), \\ (\text{T-S, S-DD, DD-T}), \\ (\text{T-DD, DD-D, D-T}), \\ (\text{T-D, D-T, T-D}), \\ (\text{T-S, S-T, T-S}) \end{bmatrix}$$

However, this $\text{F} - \text{Major}$ one is not in the quaternary key-determining chord progression candidates. It is the discarded.
For repeating chords, we just skip them. For example: $C - C - C - C - F - F - G - G - C - C$ has function key tuples list guess:

$$\mathbf{g} = \begin{bmatrix} [(\text{STAY}, \text{UNK\_KEY})], \\ [(\text{STAY}, \text{UNK\_KEY})], \\ [(\text{STAY}, \text{UNK\_KEY})], \\ [(\text{STAY}, \text{UNK\_KEY})], \\ [(\text{D-T}, \text{F}), (\text{T-S}, \text{C}), (\text{DD-D}, \text{Bb}), (\text{DD-D}, \text{a\#})], \\ [(\text{STAY}, \text{UNK\_KEY})], \\ [(\text{S-D}, \text{C}), (\text{T-DD}, \text{F})], \\ [(\text{STAY}, \text{UNK\_KEY})], \\ [(\text{D-T}, \text{C}), (\text{T-S}, \text{G}), (\text{DD-D}, \text{F}), (\text{DD-D}, \text{f})], \\ [(\text{STAY}, \text{UNK\_KEY})], \\ [(-, \text{UNK\_KEY})] \end{bmatrix}$$

When doing the quaternary key-determining chord progression detection, we just skip the $[(\text{STAY}, \text{UNK\_KEY})]$, and then choose the rests. In this case, the selection range at the first step is

$$\mathbf{g}=\begin{bmatrix}[(\mathrm{STAY},\mathrm{UNK\_KEY})],\\ [(\mathrm{STAY},\mathrm{UNK\_KEY})],\\ [(\mathrm{STAY},\mathrm{UNK\_KEY})],\\ [(\mathrm{STAY},\mathrm{UNK\_KEY})],\\ [(\text{D-T},\mathrm{F}),(\text{T-S},\mathrm{C}),(\text{DD-D},\mathrm{Bb}),(\text{DD-D},\mathrm{a\#})],\\ [(\mathrm{STAY},\mathrm{UNK\_KEY})],\\ [(\text{S-D},\mathrm{C}),(\text{T-DD},\mathrm{F})],\\ [(\mathrm{STAY},\mathrm{UNK\_KEY})],\\ [(\text{D-T},\mathrm{C}),(\text{T-S},\mathrm{G}),(\text{DD-D},\mathrm{F}),(\text{DD-D},\mathrm{f})],\\ [(\mathrm{STAY},\mathrm{UNK\_KEY})],\\ [(-,\mathrm{UNK\_KEY})]\end{bmatrix}$$

Now, everything about quaternary key-determining chord progression detection is demystified. After this, our old example is reduced into

$$\mathbf{g}=\begin{bmatrix}[(\mathrm{STAY},\mathrm{UNK\_KEY})],\\ [(\text{T-S},\mathrm{C})],\\ [(\text{S-D},\mathrm{C})],\\ [(\text{D-T},\mathrm{C})],\\ [(-,\mathrm{UNK\_KEY})]\end{bmatrix}$$

Now, there is no need to do reduction. The only things left is to rewrite the first and the last "UNK_KEY". This task is done by the function $\mathrm{visit\_nearby\_deterministic\_key}(i)$ and $\mathrm{align\_chord\_with\_detected\_keys}(\mathbf{g})$. It parses all components in $\mathbf{g}$ which are still in "UNK_KEY" or still have multiple function-key guesses. For each "undetermined function-key tuple list", it visits the nearby function-key tuple lists and check whether there is a list with determined key and in length one; if so, it checks whether the chord is in the corresponding key; if so, it inherits that key into the undetermined function-key tuple list.
In this toy example, we check whether the first C chord is in $\mathrm{C-Major}$ key; yes, it is; then, rewrite $\mathbf{g}$ into

$$\mathbf{g}=\begin{bmatrix}[(\mathrm{STAY},\mathrm{C})],\\ [(\text{T-S},\mathrm{C})],\\ [(\text{S-D},\mathrm{C})],\\ [(\text{D-T},\mathrm{C})],\\ [(-,\mathrm{UNK\_KEY})]\end{bmatrix}$$

Then, we check whether the last C chord is in $\mathrm{C-Major}$ key; yes, it is; then, rewrite $\mathbf{g}$ into

$$\mathbf{g}=\begin{bmatrix}[(\mathrm{STAY},\mathrm{C})],\\ [(\text{T-S},\mathrm{C})],\\ [(\text{S-D},\mathrm{C})],\\ [(\text{D-T},\mathrm{C})],\\ [(-,\mathrm{C})]\end{bmatrix}$$

Finally, we prune $\mathbf{g}$ into: $[(\mathrm{T},\mathrm{C}),(\mathrm{T},\mathrm{C}),(\mathrm{S},\mathrm{C}),(\mathrm{D},\mathrm{C}),(\mathrm{T},\mathrm{C})]$, finishing the whole procedure. This final function also translates binary function into unary function. For uncertain binary functions, it re-checks the position of the chords in its corresponding key as detected; if so, it assigns the unary function with the position detected.

For simplicity, we skipped the introduction of ternary and binary key-determining chord progressions. They are considered in the program. If interested, readers are free to check the code.
(Hint: there are binary key-determining chord progressions when chords have "functioning notes", like $G^9 \to C^7_\Delta$.)

To further explain the algorithm, we provide an advanced example: key analysis of chord progression of "Blue Bossa".

An Example

The chord progression of Blue Bossa is

$$\mathbf{c} = \begin{bmatrix} C^-, C^-, F^-, F^-, \\ D_{\text{dim}}, G, C^-, C^-, \\ E_{\text{b}}^{-7}, A_{\text{b}}^7, D_{\text{b}\Delta}^7, D_{\text{b}\Delta}^7, \\ D_{\text{dim}}, G, C^-, C^- \end{bmatrix}$$

- Get the binary chord progression function-key guesses

$$\mathbf{g} = \begin{bmatrix} [(\text{STAY}, \text{UNK\_KEY})] & [(\text{D-T}, \text{f}), (\text{T-S}, \text{c}), (\text{STAY}, \text{E}_\text{b})] & [(\text{STAY}, \text{UNK\_KEY})] & [(\text{T-DD}, \text{F}), (\text{T-DD}, \text{f})] \\ [(\text{S-D}, \text{c})] & [(\text{D-T}, \text{c})] & [(\text{STAY}, \text{UNK\_KEY})] & [\emptyset] \\ [(\text{S-D}, \text{D}_\text{b})] & [(\text{D-T}, \text{D}_\text{b})] & [(\text{STAY}, \text{UNK\_KEY})] & [\emptyset] \\ [(\text{S-D}, \text{c})] & [(\text{D-T}, \text{c})] & [(\text{STAY}, \text{UNK\_KEY})] & [(\text{STAY}, \text{UNK\_KEY})] \end{bmatrix}$$

- Do quaternary key-determining chord progression detection, and $\mathbf{g}$ remains unchanged.
- Do ternary key-determining chord progression detection together with $\text{visit\_nearby\_deterministic\_key}$, we have

$$\mathbf{g} = \begin{bmatrix} [(\text{STAY}, \text{UNK\_KEY})] & [(\text{D-T}, \text{f}), (\text{T-S}, \text{c}), (\text{STAY}, \text{E}_\text{b})] & [(\text{STAY}, \text{UNK\_KEY})] & [(\text{T-DD}, \text{F}), (\text{T-DD}, \text{f})] \\ [(\text{S-D}, \text{c})] & [(\text{D-T}, \text{c})] & [(\text{-}, \text{c})] & [(\text{-}, \text{c})] \\ [(\text{S-D}, \text{D}_\text{b})] & [(\text{D-T}, \text{D}_\text{b})] & [(\text{-}, \text{D}_\text{b})] & [(\text{-}, \text{D}_\text{b})] \\ [(\text{S-D}, \text{c})] & [(\text{D-T}, \text{c})] & [(\text{-}, \text{c})] & [(\text{STAY}, \text{UNK\_KEY})] \end{bmatrix}$$

- Rewrite “STAY”s:

$$\mathbf{g} = \begin{bmatrix} [(\text{STAY}, \text{f}), (\text{STAY}, \text{c}), (\text{STAY}, \text{E}_\text{b})] & [(\text{D-T}, \text{f}), (\text{T-S}, \text{c}), (\text{STAY}, \text{E}_\text{b})] & [(\text{STAY}, \text{F}), (\text{STAY}, \text{f})] & [(\text{T-DD}, \text{F}), (\text{T-DD}, \text{f})] \\ [(\text{S-D}, \text{c})] & [(\text{D-T}, \text{c})] & [(\text{-}, \text{c})] & [(\text{-}, \text{c})] \\ [(\text{S-D}, \text{D}_\text{b})] & [(\text{D-T}, \text{D}_\text{b})] & [(\text{-}, \text{D}_\text{b})] & [(\text{-}, \text{D}_\text{b})] \\ [(\text{S-D}, \text{c})] & [(\text{D-T}, \text{c})] & [(\text{-}, \text{c})] & [(\text{STAY}, \text{UNK\_KEY})] \end{bmatrix}$$

- Do $\text{align\_chord\_with\_detected\_keys}(\mathbf{g})$:

$$\mathbf{g} = \begin{bmatrix} [(\text{STAY}, \text{c})] & [(\text{T-S}, \text{c})] & [(\text{STAY}, \text{c})] & [(\text{STAY}, \text{c})] \\ [(\text{S-D}, \text{c})] & [(\text{D-T}, \text{c})] & [(\text{-}, \text{c})] & [(\text{-}, \text{c})] \\ [(\text{S-D}, \text{D}_\text{b})] & [(\text{D-T}, \text{D}_\text{b})] & [(\text{-}, \text{D}_\text{b})] & [(\text{-}, \text{D}_\text{b})] \\ [(\text{S-D}, \text{c})] & [(\text{D-T}, \text{c})] & [(\text{-}, \text{c})] & [(\text{STAY}, \text{c})] \end{bmatrix}$$

- Do $\text{prune\_function\_key\_tuples\_list}(\mathbf{g})$:

$$\text{return} \begin{bmatrix} (\text{T}, \text{c}) & (\text{T}, \text{c}) & (\text{S}, \text{c}) & (\text{S}, \text{c}) \\ (\text{S}, \text{c}) & (\text{D}, \text{c}) & (\text{T}, \text{c}) & (\text{T}, \text{c}) \\ (\text{S}, \text{D}_\text{b}) & (\text{D}, \text{D}_\text{b}) & (\text{T}, \text{D}_\text{b}) & (\text{T}, \text{D}_\text{b}) \\ (\text{S}, \text{c}) & (\text{D}, \text{c}) & (\text{T}, \text{c}) & (\text{T}, \text{c}) \end{bmatrix}$$

Chord Extension Algorithm

Suppose we are given a chord progression $[C, D, F, G, C]$, how to extend it into jazz-style chords?
A musician will immediately give a result $[C_\Delta^7, D^7, F_\Delta^7, G^7, C_\Delta^7]$. But how to automize this procedure?
First, we should have the capability to detect chord symbols. Suppose this is done.
Next, we should have the capability to detect chord functions and keys. Suppose this is done.
Finally, we have to implement the function

$$\text{new\_chord} = \text{extend\_chord}(\text{chord\_type}, \text{chord\_root}, \text{key})$$

The chord extension rules rely on chord type, chord root and key. We first define a set of chord extension rules for each chord quality (major, minor, dominant, suspended, augmented, diminished), where each chord-root-to-key-root interval from 0 to 11 are considered. Then, specifying chord-root-to-key-root interval (chord position), we can extend the given chord quality to many extension choices, which are all considered in the rules.

Check the code for relevant rules.