In situ functional dissection of RNA *cis*-regulatory elements by multiplex CRISPR-Cas9 genome engineering

GenERA pipeline overview
Author: Quentin Ferry, Last update 20170704

Disclaimer: This document contains a step-by-step explanation of the core GenERA pipeline used to process and analyze the dataset presented in this study. References to the relevant source code (filenames) are given along the explanation as well as sample data to run the codes on. All source codes have been annotated to facilitate their use. We also included in this overview document the header present at the top of each script, which describes the expected INPUTS, FUNCTIONS, and OUTPUTS of the given code. Note that before running some of the scripts below, the user path pointing to the project folder must be defined as indicated. Additionally, except for the combinatorial analysis (**GenERA_combinatorial.R**), scripts must be run sequentially as the output of one is often needed as input in the next script.

Summary of the pipeline: The GenERA pipeline aims to understand the impact of genomic deletions on transcript abundance. To do so the gDNA and cDNA from edited cells are subjected to targeted NGS (see Method section), meaning that the same amplicon (genomic sequence encompassing the edited region) is analyzed in both gDNA and cDNA. The GenERA pipeline takes as input the pair-end reads, stored as .SAM files, from the gDNA and cDNA library. Editing by CRISPR:Cas9 causes genomic deletions, which vary in size and coverage across the amplicon. The aim of the pipeline is to extract from the sequencing reads all deletions, sort them by unique deletion patterns (UDP), and match them between gDNA and cDNA libraries. By comparing the read count ratio between cDNA and gDNA libraries for a given unique deletion pattern (UDP) to the similar ratio for the non-edited amplicon, it is possible to assess the effect of each genomic ablation on transcript level.

Folder architecture:
The parent/project folder is *GenERA_GitHub* and contains the following subfolders:

- *CODE*: contains all the scripts described in this document
- *DATA*: contains all the .SAM files as well as data generated by the GenERA pipeline.
- *FIGURES*: contains the figures generated by the GenERA pipeline.
- *META*: contains metadata helping the analysis. Most metadata are provided by the user, others are generated by the GenERA pipeline.

Programming language: The core GenERA pipeline is written in Python(https://www.python.org) and R(https://cran.r-project.org). Users should install the corresponding compilers before attempting to run the scripts below.

Step 1 – Determining ROI for a given amplicon

Step 1.1 – Computing nucleotide deletion profile from sequencing data: This assumes that paired-end reads from gDNA and cDNA libraries are given for the amplicon. In order to run the script, we provided gDNA and cDNA *.SAM* files for the *crok* amplicon (see *GenERA_GitHub/DATA/SAM*). See manuscript for experimental details on how those files were generated.

For each amplicon both gDNA and cDNA *.SAM* files are used to count the number of reads bearing a deletion at a particular nucleotide along the amplicon reference genome. Each read is reconstructed using its corresponding start point and CIGAR string, as provided by the *.SAM* file, and mapped onto the reference corresponding genome. Subsequently the system counts the number of reads N_i which have a deletion at nucleotide i. At the end of this process we obtain a vector containing $N_1 \ldots N_k$ where K is the length of the amplicon. This type of data is referred to as nucleotide deletion profile in the rest of this document. This is generated with the ***GenERA_1.py*** script which takes as input the *.SAM* files and a user defined input.csv file. It is critical that the format of input.csv is respected otherwise the system will not be able to extract the relevant information from it. Each row corresponds to a given amplicon and has 7 entries: (1) file name of the gDNA *.SAM* file (name should be of the format yourName.sam.gDNA); (2) file name of the cDNA .SAM file (name should be of the format yourName.sam.cDNA); (3) Reference genome sequence used for the alignment; (4-7) coordinates of MRE and seed nucleotides (replace by 0,0,0,0 if not analyzing MREs).

NOTE: This script requires ***GenERA_functions.py*** as dependency. Both ***GenERA_1.py*** and ***GenERA_functions.py*** can be found in *GenERA_GitHub/CODE/.* input.csv can be found in *GenERA_GitHub/META/.* The outputs are as follow: nucleotide_deletion_pattern.csv and gDNA_cDNA_pairs.csv which contain the nucleotide deletion pattern for all amplicons listed in input.csv and file information, respectively. Finally, this script needs to be executed from its parent folder.

GenERA_1.py (header)

```
# ||||||||||||||||||||||||||||||||||||||||||||||||||||||||||| GenERA PIPELINE

# NAME: GenERA_1.py

# OPERATIONS:
# - reads gDNA and cDNA SAM files for a list of amplicons provided in "./META/input.csv" file
# - maps read deletions using the CIGAR
# - counts the number of reads having a deletion at nucleotide: nucleotide deletion profile
# - saves in "./META/nucleotide_deletion_pattern.csv"

# INPUT:
# - "./META/input.csv": contains a line for each amplicons:
# (format of each line) gDNA SAM file, cDNA SAM file, genome sequence, seed start, seed end,MRE
start, MRE end

# OUTPUT:
# - "./META/nucleotide_deletion_pattern.csv": contains a table with the following columns
#   - Sample type (gDNA, cDNA)
#   - position (nucleotide along the reference genome)
#    - deletion stack (one column per amplicon): counts the number of reads with deletion at
nucleotide i (specified by the position column).

# - "./META/gDNA_cDNA_pairs.csv": contains meta information for each amplicon
#   - Name
#   - sam_gDNA
#   - sam_cDNA
#   - ref_genome
#   - read_count_gDNA
#   - read_count_cDNA
#   - seed_start
#   - seed_end

# DEPENDENCIES: GenERA_functions.py

# |||||||||||||||||||||||||||||||||||||||||||||||||||||||||||
```

Step 1.2 – Determining ROI from nucleotide deletion profiles: The aim of the GenERA pipeline is to group all read sharing the same deletion, or unique deletion pattern (UDP). Certain reads being shorter, meaning they do not cover the entire amplicon sequence, might appear as if they have two deletions: the biologically relevant deletion caused by CRISPR:Cas9 and a technical deletion caused by the read being shorter than the amplicon sequence. In order to group together reads sharing identical Cas9 deletions the system is design so as to compare reads only in a region of interest (ROI), which is defined as the smallest region encompassing all biologically relevant UDPs. To derive the ROI, we created a custom made peak calling algorithm that determines the coordinate of the ROI relative to the amplicon based on the nucleotide deletion profile (***GenERA_2.R***). The nucleotide deletion profile (see step 1.1) is first smoothed 10 times by taking at each nucleotide *i* the average values of nucleotides *i-1*, *i* and *i+1*. The smoothed curve is then fed to a peak detector: the first derivative of the curve is computed by averaging left and right variations; then the values from both the smoothed curve and first derivative are used to extract blobs of deletions (segments of the amplicon which encompass a group of deletions). The ROI for a given amplicon is determined as the boundaries the largest blob returned by the peak calling algorithm. Note that a ROI is computed for gDNA and cDNA libraries separately and the amplicon ROI is the union of these two ROIs.

NOTE 1: ***GenERA_2.R*** needs to run in R from the parent directory. User will need to set the working directory (first line of code at the top of the file) to the parent directory. The code takes as input nucleotide_deletion_pattern.csv (see previous step) and output a graph for each amplicon showing the smoothed nucleotide deletion patterns for gDNA and cDNA libraries as well as their predicted ROI (see *./FIGURES/ROI/*). The final ROI is saved in the csv file ROI.csv.

NOTE 2: ROI.csv will be used downstream for UDP characterization (see below). The user can overwrite the file with manually chosen ROI coordinates when required. If for example Cas9 is used with two guide RNAs targeting two distinct regions of the amplicon, the software would most likely identify two deletions blobs rather than one encompassing all deletion events. The ROI automatically chosen by the software will be the largest of these peaks.

GenERA_2.R **(header)**

```
# |||||||||||||||||||||||||||||||||||||||||||||||||||||||||||| GenERA PIPELINE

# NAME: GenERA_2.R

# FUNCTION:
# - reads nucleotide deletion profile files and determines a Region Of Interest (ROI) to work with
for each amplicon (deletion peak calling)

# INPUT:
# - "./META/nucleotide_deletion_pattern.csv": contains the deletion counts for gDNA and cDNA at
single nucleotide resolution across the referecne genome
# (see output of mreCRISPR_1.py)

# OUTPUT:
# - "./META/ROI.csv": contains for each amplicon the amplicon ROI coordinates containing both gDNA
and cDNA peaks.
# One row per amplicon, columns are as follow:
# - name: name of the amplicon
# - start: left coordinate of the zone (in the genome reference)
# - end: left coordinate of the zone (in the genome reference)
# - divergence: measure of overlap between gDNA and cDNA deletion peak

# DEPENDENCIES: none

# ||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||
```

Step 2 - Extracting and cropping read sequences to the ROI for each amplicon

Once the ROI for a given amplicon is known the system goes back to the original *.SAM* files and extract the alignment information of each read in the ROI. This is done by: (1) reading the reads in gDNA and cDNA sequencing libraries; (2) reconstituting the read sequence; (3) cropping the read sequence to the ROI (see step 1.2); (4) Generating a cropped CIGAR string which describes the read alignment (match, mismatch, deletion, etc.) in the ROI. Once this is completed, the system builds a repertoire of unique CIGARs and count for each one the number of reads in gDNA and the number of reads in cDNA which share this alignment. Note that some CIGARs will be shared between gDNA and cDNA repertoires while some will be only present in one or the other. This is performed by the ***GenERA_3.py*** script.

NOTE: This script requires ***GenERA_functions.py*** as dependency. Both ***GenERA_3.py*** and ***GenERA_functions.py*** can be found in *GenERA_GitHub/CODE/.* Inputs for this script, namely gDNA_cDNA_pairs.csv and ROI.csv, can be found in *GenERA_GitHub/META/*. The outputs are .csv file containing the GenERA data for each amplicon. Those are saved in *./DATA/CSV/A_RAW_CSV_FILES/*.

GenERA_3.py (header)

```
# |||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||| GenERA PIPELINE

# NAME: GenERA_3.py

#  FUNCTION:  For  each  amplicon  extract  from  SAM  files  wild-type  and  all  UDP  (unique  deletion
patterns) in the zone
#  -  Using  zone  coordinates,  extract  for  each  amplicon  the  list  of  unique  cigars  and  their
corresponding read counts
# for both gDNA and cDNA
# - cDNA and gDNA cigars are then matched if possible and compiled into a common file

# INPUT:
# - "./META/gDNA_cDNA_pairs.csv": see GenERA_1.py output
# - gDNA SAM files
# - cDNA SAM files
# - "./META/ROI.csv": ROI delineation for each amplicon see GenERA_2.R output

# OUTPUT:
# - './DATA/CSV/A_RAW_CSV_FILES/'+ L_file_pairs[i][0] + '_GenERA.csv': csv file cataloguing the cDNA
and gDNA cigars and their respective read counts (matched when possible)

# DEPENDENCIES: mreCRISPR_functions.py

# ||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||
```

Step 3 – Data formatting, filtering and UDP pairing between libraries:
Step 3.1 – Reformatting: Data generated by the Python script are first reformatted to be compatible with the remainder of the pipeline (R scripts). This is carried out by the ***GenERA_4.R*** script.

NOTE: ***GenERA_4.R*** needs to be run in R from the parent directory. User will need to set the working directory (first line of code at the top of the file) to the parent directory. This script takes as input the *_GenERA.csv files generated with ***GenERA_3.py*** and output the reformatted files in *./DATA/CSV/A_REFORMATED_CSV_FILES/*.

GenERA_4.R (header)

```
# |||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||| GenERA PIPELINE

# NAME: GenERA_4.R

# FUNCTION:
#  - reformat  output  from  GenERA_3.py  (add  one  extra  column)  to  be  compatible  with  downstream  R
scripts

# INPUT:
# - output files from GenERA_3.py (raw CSV files)

# OUTPUT:
# - reformated output from GenERA_3.py (reformated CSV files)

# DEPENDENCIES: none

# ||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||
```

Step 3.2 – UDP matching across libraries: Taking the registered reads cropped to the ROI (see step 2) an extra step of filtering (quality control) is applied before characterization of the UDP repertoire. First, incomplete reads (reads showing deletion at the edges of the ROI) are filtered out.

These are removed as it is impossible to know the extent of the deletion pass the ROI. A second filtering step is applied aimed at removing hyper-mutated reads (we use a conservative cutoff of > 10% of the ROI mutated). Reads are then converted into a binary sequence which represents the deletion status of nucleotide along the ROI. We refer to this binary signature as a unique deletion pattern (UDP). Note that in this process, mismatches caused by PCR and sequencing artefacts are reverted to the wild-type sequence, increasing the number of reads that share the same UDP. As an example if the wild-type sequence is AGCTAGCTGATGCTAG, and the analyzed read is AG**G**TAG------CTAG, then the binary conversion of this read would be XXXXXX******XXXX where X stands for match and * stands for deleted. Using this binary signature or UDP, the system computes the UDP repertoire of the amplicon and counts for each UDP the number of gDNA reads and cDNA reads having this binary deletion pattern. This is performed by the ***GenERA_6.R*** script.

NOTE: ***GenERA_6.R*** needs to be run in R from the parent directory. User will need to set the working directory (first line of code at the top of the file) to the parent directory. This script requires the dependency ***GenERA_functions.R*** which contains all functions for the R scripts. Inputs are: (1) *_GenERA.csv files from the previous step found in *./DATA/CSV/A_REFORMATED_CSV_FILES/*; (2) GENOMES.csv (found in *./META/*) which list the reference genome sequence of each amplicon. Merged *_GenERA.csv files produced for each amplicon are stored in *./DATA/CSV/A_FINAL_CSV_FILES/*. Additionally, a merging_log.csv file is created in the META folder which gives stats on the process.

GenERA_6.R (header)

```
# |||||||||||||||||||||||||||||||||||||||||||||||||||||||||||| GenERA PIPELINE

# NAME: GenERA_6.R

# FUNCTION:
# - remove the incomplete reads (reads with dilution on 5' or 3' of the ROI)
# - remove all cigars with too many mutations
# - compute DATA_merge = data.frame(seq_zone_XDX,
#                                 seq_seed_XDX,
#                                 count_gDNA,
#                                 count_cDNA,
#                                 seed_start,
#                                 seed_end,
#                                 zone_length)
# - remove unmatched
# - add count_min, deletion (boolean), deletion_seed (boolean) columns
# - compute UDP score
# - compute normalised UDP score (UNS)
# - boolean check if the deletion is contained or not within the MRE
# - check for de-novo MRE: (now done in mreCRISPR_7.R)
#   - map back the UDP on the reference genome
# - get genome sequence post deletion
# - search for and flag creation or removal of miRNA seed from top10 miRNA in S2+ cells

# INPUT:
# - output from GenERA_3.py (created with GenERA_4.R) in ./DATA/CSV/A_REFORMATED_CSV_FILES/
# - GENOMES.csv file which contains the reference genome sequence of the amplicon

# OUTPUT:
# - csv of DATA_merge in ./DATA/CSV/A_FINAL_CSV_FILES/

# DEPENDENCIES: GenERA_functions.R

# ||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||
```

NOTE: Additional statistics on the merging process can be computed using ***GenERA_5.R*** (see below). ***GenERA_5.R*** needs to be run in R from the parent directory. User will need to set the working directory (first line of code at the top of the file) to the parent directory. This script requires the dependency ***GenERA_functions.R*** which contains all functions for the R scripts. Inputs for this script are the *_GenERA.csv files found in *./DATA/CSV/A_REFORMATED_CSV_FILES/*. The ouput is a .csv file global_stats.csv saved in the META folder and contains all the statistics listed in the header below.

GenERA_5.R (header)

```
# ||||||||||||||||||||||||||||||||||||||||||||||||||||||||||| GenERA PIPELINE

# NAME: GenERA_5.R

# FUNCTION:
# - load reformated file
# - STATS: total_gDNA_count
# - STATS: total_cDNA_count
# > remove all cigars in zone with '-' (searched in sequence in the zone)
# - STATS: total_gDNA_count
# - STATS: total_cDNA_count
#
# - STATS: seed_start_in_zone
# - STATS: seed_end_in_zone
# - STATS: zone_length
#
# - STATS: perfect_wt_gDNA_count (=0 if no wild-type cigar)
# - STATS: perfect_wt_cDNA_count (=0 if no wild-type cigar)
#
# > remove cigars with too many mutations
# - compute mutation frequency of all cigars
# - keep only mutation frequency < 0.1 cigars
# - STATS: post_mutated_valid_gDNA_count
# - STATS: post_mutated_valid_cDNA_count
# - STATS: nb_UDP_pre_merge
#
# > merge similar cigars after ignoring mutations
# - convert seq_zone and seq_seed to X***X profiles
# - pair reads based on seq_zone_XDX
# - compute DATA_merge (seq_zone_XDX,
#                       seq_seed_XDX,
#                       count_gDNA,
#                       count_cDNA,
#                       seed_start,
#                       seed_end,
#                       zone_length)
# - remove all unpaired UDPs
# - STATS: nb_UDP_post_merge
# - STATS: wt_gDNA_count
# - STATS: wt_cDNA_count
#
# > UDP stats
# - STATS: nb_deletion_UDP_post_merge
# - STATS: deletion_UDP_post_merge_gDNA_count
# - STATS: deletion_UDP_post_merge_cDNA_count
#
# - STATS: nb_deletion_out_seed_UDP_post_merge (outside seed)
# - STATS: deletion_out_seed_UDP_post_merge_gDNA_count
# - STATS: deletion_out_seed_UDP_post_merge_cDNA_count
#
# - STATS: nb_deletion_seed_UDP_post_merge (touch seed)
# - STATS: deletion_seed_UDP_post_merge_gDNA_count
# - STATS: deletion_seed_UDP_post_merge_cDNA_count
#
# - STATS: nb_full_deletion_seed_UDP_post_merge (complete seed deletion)
# - STATS: full_deletion_seed_UDP_post_merge_gDNA_count
# - STATS: full_deletion_seed_UDP_post_merge_cDNA_count
#
# - STATS: nb_deletion_mre_only_UDP_post_merge ***
# - STATS: deletion_mre_only_UDP_post_merge_gDNA_count
# - STATS: deletion_mre_only_UDP_post_merge_cDNA_count

# INPUT:
# - output from GenERA_3.py (reformatted with GenERA_4.R)

# OUTPUT:
# - stats.csv file

# DEPENDENCIES:
# - GenERA_functions.R

# |||||||||||||||||||||||||||||||||||||||||||||||||||||||||||
```

Step 3.3 – UNS calculations: Once the data is merged (identification of the UDP repertoire) a UDP score is calculated for each UDP as the ratio between the cDNA read counts and gDNA read counts bearing this deletion. Using the score of the wild-type UDP (no deletion), a UDP normalized score (UNS) is calculated by dividing the UDP score with the wild-type score. This is also performed in the ***GenERA_6.R*** script.

Step 4 – Look for potential confounding factors
Using ***GenERA_7.R*** it is possible to remove two types of potentially confounding factors: for example, if the user is interested in examining the regulatory potential of *element x*, Cas9 will be used to generate a repertoire of UDPs overlapping with this element. If a UDP overlaps with this element as well as an additional known or unknown regulatory element, then the change in transcript levels cannot be solely attributed to the element of interest. Also if the user has a list of coordinates for predicted regulatory elements in the region (see predicted_RREs.csv) the GenERA pipeline allows to flag UDPs affecting these elements so that they can be removed in a downstream analysis. Additionally, Cas9 caused deletions can lead to the formation of de-novo MREs for highly expressed miRNAs in the cell line of interest. Using GENOMES.csv and TOP10_miRNA.csv the GenERA pipeline makes it possible to search UDPs for the creation of such elements.

NOTE: ***GenERA_7.R*** needs to be run in R from the parent directory. User will need to set the working directory (first line of code at the top of the file) to the parent directory. This script requires the dependency ***GenERA_functions.R*** which contains all functions for the R scripts. In addition to *_GenERA.csv files generated by ***GenERA_6.R***, inputs for this script include the following three meta files: GENOME.csv described earlier; predicted_RREs.csv contains coordinates of predicted regulatory elements along the different amplicons; TOP10_miRNA.csv provides the names, seed sequences, and expression levels of the most expressed miRNAs in S2 cells.

GenERA_7.R (header)

```
# |||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||| GenERA PIPELINE

# NAME: GenERA_7.R

# FUNCTION:
# - check all UDP footprints for overlap with predicted regulatory elements.
# - check all UDP footprints for de-novo deletion/creation of MREs for top 10 S2 miRNAs (including
miR-184)

# INPUT:
# - merge UDP files from GenERA_6.R foud in ./DATA/CSV/A_FINAL_CSV_FILES/
# - ./META/GENOMES.csv contains the reference genome sequence for each amplicon
# - ./META/predicted_RREs.csv contains the name and coordinates of predicted regulatory elements for
each amplicon
# - ./META/TOP10_miRNA.csv contains the name, seed sequence, and expression levels of the top 10
most expressed miRNA in S2 cells


# OUTPUT:
# - enhanced UDP files (stored in './DATA/CSV/A_FINAL_enhanced_CSV_FILES/')

# DEPENDENCIES: mreCRISPR_functions.R

# ||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||
```

Explanation of the *_GenERA.csv file format: The GenERA pipeline produces for each amplicon a *_GenERA.csv file which contains information on the Cas9-generated UDP repertoire. For each row (UDP) the following information is given (columns):

- seq_zone_XDX: binary deletion pattern (UDP) over the ROI
- seq_seed_XDX: binary deletion pattern restricted to the targeted MRE seed region
- count_gDNA: Number of gDNA reads with this UDP
- count_cDNA: Number of cDNA reads with this UDP
- seed_start/end: coordinate of the targeted MRE seed sequence
- zone_length: length of the ROI
- count_min: minimum value between count_gDNA and count_cDNA
- deletion: Boolean indicating if the UDP contains a deletion. “False” value indicate wild-type UDP.
- deletion_seed: Boolean indicating if the UDP overlap with the target MRE seed
- score: ratio between cDNA and gDNA read counts

- score_norm_divide: score divided by wild-type score
- score_norm_minus: score minus wild-type score
- MRE start/end: coordinate of the target MRE in the amplicon
- mre_only: Boolean indicating if the UDP is contained within the target MRE
- DNM: reports on the formation or ablation of seed sequences for T10 miRNAs
- FeatureDeletion: reports on ablation of predicted regulatory elements
- DNM_exclu: Boolean deciding if the read is to be excluded in downstream analysis based on ablation of foreign elements

Step 5 – Combinatorial editing patterns:
GenERA analysis can be extended to the study of combinatorial effects engendered by concomitant ablation of several regulatory elements. In practice, this enables the user to test if the simultaneous ablation of element 1 and element 2 has an additive effect compared to the deletion of one or the other in isolation. To this end, the GenERA pipeline allows the user to define multiple elements across the amplicon by providing their coordinates along the reference genome (see ***GenERA_combinatorial.R***). For example, we assume that four elements have been listed by the user. GenERA can then assign to each deletion in the UDP repertoire a combinatorial editing patter (CEP) which contains the deletion status of the four annotated elements. For example deletion of element 1 and 2 will give the CEP '--XX' where '-' stands for deleted and 'X' stands for intact elements. One can then characterize the effect of a particular CEP on transcript abundance by examining the UNS distribution of its UDPs. From there it is also possible to use GenERA (***GenERA_combinatorial.R***) to search all the cases of combinatorial editing which can be describe as the combination of two simpler editing events. For example '-X-X' can be seen as '-XXX' + 'XX-X'. GenERA will search the list of CEPs to find such relationships and let the user compare the UNS distribution of the combinatorial editing (parent) with the distribution of the children CEPs. This is done using the code ***GenERA_combinatorial.R*** describe below.

NOTE: ***GenERA_combinatorial.R*** needs to be run in R from the parent directory. User will need to set the working directory (first line of code at the top of the file) to the parent directory. This script requires the dependency ***GenERA_functions.R*** which contains all functions for the R scripts. In order to run this script, we provide the data_combinatorial_GenERA.csv file which contains data from combinatorial Cas9 editing of a single amplicon. Outputs for this script are as follow: (1) zone_boxplot.csv which contains UNS values for UDPs restrained to user defined zone 1, zone 2, or editing zone 1 and zone 2 concomitantly; (2) cepComb* files which contain the output of the parent/children analysis.

GenERA_combinatorial.R (header)

```
# |||||||||||||||||||||||||||||||||||||||||||||||||||||||||||| GenERA PIPELINE

# NAME: GenERA_Combinatorial.R

# FUNCTION:
# - remove low depth UDPs
# - amend data with information regarding the length of the deletion and wether or not it is
continuous
# - Establish list of CEP and filter out CEP which have less than N representative UDPs available
(here we chose N = 10)
# - Provide code to compare the UNS distribution between UDP mapping to zone 1 (user defined), zone
2 (user defined) or both. With possibility for quantile filtering
# - Test the additive property of combinatorial editing by finding all CEP trios where CEP3 (parent)
is a combination of CEP1 and CEP2 (children). Extracts corresponding UNS for all 3 CEPs.

# INPUT:
# - '../DATA/CSV/B_combinatorial/data_combinatorial_GenERA.csv', csv data file as produced by
GenERA_6.R

# OUTPUT:
# - 'zone_boxplot.csv' contains the result of the two zone analysis
# - './export/cepComb_---x---_1110000_0000111_.csv' series of .csv files containing the result of
the parent children combinatorial analysis

# DEPENDENCIES:
# - mreCRISPR_functions.R

# ||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||
```

Closing remark: The additional codes used in our study to produce the final plots have not been included in this release in the interest of simplicity and functionality. Nevertheless, those scripts are available upon request addressed to the corresponding author.