Annex 2: Algorithms for block event inference, detailed description

Notations:

Input data:

- S is the species tree
- $\{T\}$ is the collection of all the pangenome's gene family trees
- R is a replicon, i.e. a vector of genes of size r

Gene tree indexation:

- G_i is the i-th gene of R gene, and T_i the tree of its gene family
- $t(n_i)$ is the vector of nodes on the lineage of T_i leading to the tip representing G_i, i.e. the path between the gene tree root and the tip
- $n_{i,j}$ is the j-th node on $t_{i,}$ counting from the tip representing G_i (that is $n_{i,0}$), i.e. $n_{i,j} = t(n_i)[j]$

ODT event indexation:

- gene tree nodes are annotated with events, and given the event collection $\{E\}$, the map Ev links a node n with the event $E_n = \mathrm{Ev}(n, \{E\})$
- $E_{i,j}$ is the event associated to $n_{i,j}$
- m is a node in S representing an ancestral species
- $L(X) = \{m\}$ is the location of the event or block event X, characterized by the set of possible nodes in S where it can have occurred

Block classes:

- B is a leaf block event, instance of class *LBlock,* characterized by the following attributes: the orderred vector of genes it gathers $(G)_B$, the collection of associated events $\{E\}_B$, and its aggregate location $L(B)$
- A is an ancestral block event, instance of class *ABlock,* characterized by the following attributes: the collection of leaf bllock events it gathers $(B)_A$, the collection of associated gene tree events $\{E\}_A$, and its aggregate location $L(A)$

Block construction (parameters):

- g is the maximum allowed gap length for leaf block event construction

Block construction (dynamic variables):

- $(A)_c$ and $(B)_c$ are the current lists of all ancestral and leaf block events, respectively
- $\{A\}_E$ is the set of ancestral blocks that are currently linked to a gene tree event E (through any of their member leaf blocks)
- *getanc* is a function such as $\{A\}_E = getanc(E)$
- P_B is the list of ancestral blocks with which leaf block B shares gene tree events, and could be putatively assigned to
- $A(B)$ is the ancestral block event to which leaf block B is currently assigned
- c is the currently opened gap size (during leaf block event construction)

Helper Functions:

```
// instanciate new blocks
def  new(LBlock, E, G) :
        set B ;
           set L(B) = L(E) ;
           set (G)B = (G) ;
           set {E}B = {E} ;
           return B ;

def  new(ABlock, B) :
        set A ;
        set (B)A = (B);
        set L(A) = L(B) ;
        set {E}A = {E}B ;
        return A ;

// add event or leaf block to leaf or ancestral block, respectively
def add(B, E, G):
       (G)B += (G)  ;
       {E}B = {E}B ∪ {E} ;
       L(B) = L(B) ∩ L(E) ;

def add(A, B):
       (B)A += (B)  ;
       {E}A = {E}A ∪ {E}B ;
       L(A) = L(A) ∩ L(B) ;

// test compatibility between two instances of the following:
// events, leaf blocks or ancestral blocks
def compat(X, Y):
       Lcompa = L(X) ∩ L(Y) ;
       return (Lcompa ≠ Ø) ;
```

```
// split block B before element i
def split(B, i):
    // create a new block B'covering the right-hand part of B
    B' = new(LBlock, {E}B[i:], (G)B[i:])
    // restrict block B to its left-hand part
    set (G)B = (G)B[1:i-1] ;
    set {E}B = {E}B[1:i-1] ;
    set L(B) = L(E1) ∩ L(E2) ∩ … ∩ L(Ei-1)
    return ( B, B' )

def merge(A1, A2):
     (B)A1 = (B)A1 ∪ (B)A2 ;
     L(A1) = L(A1) ∩ L(A2) ;
     {E}A += {E}B ;
     del A2 ;
     return A1 ;

// obtain all ancestral blocks linked to
// (i.e. currently known to include) a gene tree event E
def getanc(E, (A)c) :
       set {A}E = {}
       for A in (A)c ; do
           if (( E ∈ {E}A )) ; then {A}E += (A) ; endif
       end ;
       return {A}E ;
```

```
// resolve coordinates incompatibilities between leaf blocks
//  and linked ancestral blocks by splitting the leaf block into
// as many needed leaf blocks that are compatible with their
// respective linked ancestral blocks
def resolve(B, A):
    I = () ; // vector of indexes of events in B not compatible with A
    for i in (i >= 1; i <= len({E}B; i++) ; do
        Ei = {E}B[i]
        if ( not compat(Ei, A) ) ; then
                 I += (i) ;
        endif
    end
    for i in I ; do
        if ( i > 1) ; then
             // split block B before incompatible event Ei
             (B, B') = split(B, i) ;
             // reccursive call ; returns a vector of blocks
             // (compatible part of the original blocks always first)
             return ( B ) + resolve(B', A)
        else
             // split block at the next compatible event
             for (k >= 2; k <= len({E}B; k++) ; do
                 if ( k ∉ I ) ; then
                     (B, B') = split(B, k) ;
                     break // the for k loop
                 endif
             end
             // reccursive call ; returns a vector of blocks
             return  resolve(B', A) + ( B )
        endif
    end
    // end of reccursive call
    return ( B )
```

Algorithm 1: Construction of leaf blocks.

INPUT:

- a vector of r genes G_i orderred on a replicon R (single DNA molecule in an extant genome),
- the collection $\{T\}$ of all reconciled gene trees, of which every node n from the set $\{n\}_T$ is annotated with an event E characterized by a location L(E) in the species tree,
- a map F associating a gene G_i to its respective reconciled gene family tree T_i = F(G_i, $\{T\}$)

OUTPUT:

- a list of leaf block events $(B)_R$ for replicon R, and their respective attributes

```
// start with an empty list of leaf block events
(B)R = ()
for ((i >= 1 ; i <= r ; i++)) ; do
   // access the i-th gene on R
   Gi = R[i] ;
   // access the corresponding gene tree, the tip node representing the gene and the lineage above
   Ti = F(Gi, {T}) ;  ni  = Ti[Gi] ; ti = t(ni) ;
   // explore the lineage above
   for ((j >= 1 ; ni,j ≠ root(Ti) ; j++)) ; do
     ni,j = ti[j]
     Ei,j = Ev(ni,j, {E})
     // instantiate a new block event B with Ei,j as seed
     B = new(LBlock, Ei,j,Gi) ;
     c = 0 ;
     k = i + 1 ;
     while (( c <= g and k < r )) ; do
       // scan the right-hand neighbour genes Gk on the replicon
       Gk = R[k] ;
       Tk = F(Gi, {T}) ;  nk  = Tk[Gk] ; tk = t(nk) ;
       l = 1 ;
       while (( l <= length(tk) )) ; do
         // lineage above neighbour gene on the right Gk is explored
         nk,l = tk[l] ; Ek,l = Ev(nk,l, {E}) ;
         // test compatibility of events locations, i.e. their non-null intersection
         if ( compat(B, Ek,l) ) ; then
           add(B, Ek,l, Gk)B  ;
           c = 0 ;
           break // stop the exploration of Tk for events compatible to Ei,j
         endif
         if (( l == length(tk) )) ; then
           // no compatible event was found in lineage above Gk
           (G)B += (Gk) ; // add Gk to B as a gap gene (no refinement of L(B))
           // increment gap count
           c++ ;
         endif
       l++ ;
       end
   k++
   end
   if (( c == g )) ; then
     // there is no more compatible neighbour to add, the trail of gap genes must be trimmed from the block
     (G)B = (G)B[1:len(B)-g]
     (B)R += (B)
   endif
end
return (B)R
// end of the block event reconstruction for replicon R ; repeat for all replicons in the genome dataset
```

Algorithm 2: Construction of ancestral blocks.

INPUT:

- the list of all leaf blocks $(B)_c$

OUTPUT:

- the list of ancestral blocks $(A)_c$
- added attributes to leaf blocks, A(B) pointing to their respective ancestral blocks

```
// initiate the ancestral block list
(A)c = () ;
for (b >= 1 ; b <= len((B)c) ; b++) do ;
    B = (B)c[b] ;
    // search for already existing ancestral blocks linked to gene tree events from B
    PB = () ;
    set A(B) = Ø ;
    for E in {E}B ; do
      {A}E = getanc(E, (A)c) ; PB += {A}E ;
    end
    if ( PB ≠ Ø ) ; then
        A = new(ABlock, B) ; PB += (A) ;
        (A)c += (A) ;
    endif
    for (i >= 1 ; i <= len(PB) ; i++) ; do
        Ai = PB[i]
        // test compatibility of events locations, i.e. their non-null intersection
         if ( not compat(B, A) ) ; then
             // split block into compatible and incompatible parts
             // compatible part of the original blocks is always
             // returned as first element of the vector
             (B, B', … ) = resolve(B, A) ;
             // append new block to global list for further allocation
             (B)c += (B', … ) ;
         endif
         if ( A(B) == Ø ) ; then
             A(B) = Ai ;
         else
             A(B) = merge(Ai, AB) ;
        endif
   end
end
```