strelka2 somatic source code

1 Process input read alignment

- 1.1 Filter input read alignment flag 512 | 1024(mkdup) | 4(unmapped) | 256(secondary) | 2048(sunpplementary)
- 1.2 Tier Mapped Tier1Mapped read mapq > 20 and mate mapped and proper__pair Tier2Mapped read mapq > 0 or mate unmapped or (mate mapped not proper__pair) clearner read cigar
  1 Remove zero length alignment segments
  2 Remove pad segments
  3 Condense repeated segments
  4 Reduce adjacent insertion/deletion tags to a single pair
  5 Replace Skip-Del-Skip (NDN) pattern with single SKIP (N) segment
- 1.3 Normalize/left-shift the input read 1 collapse internal indels strelka2 source code implementation Function 1.1 collapseInsert

```
Function 1.1 collapse Indel
/// REF:   ACTGC
/// READ:  ACGC
/// CIGAR: 2M1I2D1M -> 2M1D2M
leftCollapse = 0;
rightCollapse = 0;
maxCollapse = min (insert_len, delete_len)
refRightPos = ref pos
readRightPos = read pos
while rightCollapse < maxCollapse
    isRightMatch <- read[refRightPos-rightCollapse] == refseq[refRightPos-rightCollapse]
    if isRightMatch
        rightCollapse++
    else
        break;

if rightCollapse > 0
    cagar[currentSegment].type = Match
    cagar[currentSegment].langth += rightCollapse

refLeftPos <- ref pos - delete_leng
readLeftPos <- read pos - insert_leng
while leftCollapse+rightCollapse < maxCollapse
    isLeftMatch <- read[readLeftPos+leftCollapse] == refseq[refLeftPos+leftCollapse]
    if isLeftMatch
        leftCollapse++
    else
```

```
        break;

if leftCollapse > 0
    carga[PriorMatch].type = Match
    carga[PriorMatch].length += leftCollapse

ai.priorInsertLength -= leftCollapse+rightCollapse;
ai.priorDeleteLength -= leftCollapse+rightCollapse
frist_insert <- true
first_delete <- true

for index <- indel_start to indel_end
    ps2 <- al.carga[index]
    if (ps2.type == Insert )
        ps2.length <- frist_insert == true? ai.priorInsertLength: 0;
        frist_insert <- false

    if (ps.type == Delete)
        ps2.length <- frist_insert == true? ai.priorDeleteLength: 0;
        frist_insert <- false
```

2 left shift read implement by Function 1.2 leftShiftIndel

```
// Ref  ACXXGG       ACTCGG
// READ ACTCGG       ACXXGG
// 2M2I2M -> 1M2I3M  2M2D2M _> 1M2D2M
Function 1.2 leftShiftAlignmentIndels
shift <- 0
refRightPos <- ai.refPos-1
readRightPos <- ai.readPos-1
refLeftPos <- refRightPos-ai.priorDeleteLength
readLeftPos <- readRightPos-ai.priorInsertLength
while shift < ai.priorMatchLength
    isLeftMatch <- readSeq[readLeftPos+shift] == refSeq[refLeftPos+shfit]
    isRightMatch <- readSeq[readRightPos-shift] == refSeq[refRightPos-shfit]
    if !isRightMatch && isLeftMatch
        break;
    else
        shift++
al.path[currentSegment].type = MATCH;
al.path[currentSegment].length += shift;
al.path[PriorMatchSegment].lenght -= shift
```

3 normalizeEdgeIndels handle edge indels

Function 1.3 leftEdgeIndelCollapse

```
leftEdgeIndelCollapse
```

```
if cagar.type == DELETE
    read_pos += cagar.length
if cagar.type == INSERT
    rightCollapse <- 0
    while rightCollapse < cagar.length
        isRightMatch <- read[readRightPos-rightCollapse] == ref[refRightPos-rightCollapse]
        if (!isRightMatch) reak
        rightCollapse++

rightEdgeIndelCollapse
if cagar.type == INSERT
    leftCollapse <- 0
    while leftCollapse < cagar.length
        isLeftMatch <- read[readLeftPos+leftCollapse] == ref[refLeftPos+leftCollapse]
        if (!isLeftMatch) break
        leftCollapse++
```

- 1.4 Insert Read add Alignment Indels To PosProcessor **Function 1.4** addAlignmentIndelsToPosProcessor read there is swap or simple indel by **process__swap** or **process__simple__indel** implement, have same function **finish__indel__sppr** If Indel is Novel, then initialize all class data which does not depend on read observations by **initializeAuxInfo** repeat analysis implement by **set__repeat__info**, to compute *repeatUnit*, *repeatUnitLength*, *refRepeatCount*, *indelRepeatCount*
  case 1 : Insert seq <- insert seq => insert_repeat_count
  case 2 : Delete seq <- delete seq => delete_repeat_count
  case 3 : SWAP first compute insert seq repeat, second compute delete seq repeat => insert_repeat_count and delete_repeat_count if insert seq not equal delete seq, then return

  ```
  Function get_seq_repeat_unit
  IN : seq
  OUT : repeat_unit, repeat_count
  n <- indel_seq.size()
  for i <- 1 to n -1
  is_repeat <- true
  if n % i == 0
      for j <- i; j < n; j += i
          if (indel_seq[j,...,j+k] != indel_seq[0,...,k])
              is_repeat <- false
              break;
  if (is_repeat)
      repeat_unit <- indel_seq[0,...,i]
      repeat_count <- n / i
  ```

  count repeats in contextual sequence: (indel_context_repeat_count) indel__begin__pos : indelKey.pos indel__end__pos : indelKey.pos +

indelKey.delete_length

```
indel_context_repeat_count = 0
for pos=indel_begin_pos-repeat_count; pos >=0; pos -= repeat_count
if ref[pos,...] == repeat_unit
    indel_context_repeat_count++;
for pos=indel_end_pos+repeat_count; pos < ref_end; pos += repeat_count
if ref[pos,...] == repeat_unit
    indel_context_repeat_count++;
refRepeatCount = indel_context_repeat_count + delete_repeat_count
indelRepeatCount = indel_context_repeat_count + insert_repeat_count
```

Interrupted Homopolymer Length Get the length of the longest homopolymer extending from pos when one alternate base is allowed to interrupt the homopolymer sequence. The implementation by **getInterrupted-HomopolymerLength**, indelReportInfo.interruptedHomopolymerLength = max([inde__begin_pos-1, inde__begin__pos, indel__end__pos-1, indel__end__pos].getInterruptedHomopolymerLen

```
Function getInterruptedHomopolymerLength
n1 <- 0
n2 <- 0
prior_base = ref[pos]
for pos = current_pos; pos >= 0 ; --pos
    base <- ref[pos]
    if base == prior_base
        n1++;
    else
        if prior_base = base
            n2++
        else
            break;
for pos =  current_pos+1; pos <= ref_end; ++pos
    base <- ref[pos]
    if base == prior_base
        n1++;
    else
        if prior_base = base
            n2++
        else
            break;
n3 = max(n1, n2)

n1 = 0
n2 = 0
prior_base = ref[pos]
for pos = current_pos; pos < ref_end ; ++pos
    base <- ref[pos]
    if base == prior_base
        n1++;
    else
        prior_base = base // togging on new base once
        if prior_base == base
            n2++
        else
            break;
for pos =  current_pos+1; pos <= ref_end; ++pos
    base <- ref[pos]
    if base == prior_base
        n1++;
    else
        prior_base = base // togging on new base once
        if prior_base = base
            n2++
        else
            break;
n4 = max(n1, n2)
n = max(n3, n4)
```

Function 1.5 initializeAuxInfo compute _errorRates refToIndelErrorProb, indelToRefErrorProb, candidateRefToIndelErrorProb, candidateIndelToRefErrorProb by implement **initializeAuxInfo**
case 1: simple indel, implement by code as fellow
case 2: swap indel refToIndelErrorProb <- errorRates[0][0]; indelToRefErrorProb <- errorRates[0][0]

```
repeatingPatternSize = max (repeatUnitLength, 1)
refPatternRepeatCount = max(refRepeatCount, 1)
indelPatternRepeatCount = max (indelRepeatCount, 1)

//Adjustment refPatternRepeatCount indelPatternRepeatCount
if repeatingPatternSize > 1
    repeatingPatternSize = 1
    refPatternRepeatCount = 1
    indelPatternRepeatCount = 1
else
    refPatternRepeatCount = max(15, refPatternRepeatCount)
    indelPatternRepeatCount = max(15, indelPatternRepeatCount)

refToIndelErrorProb = errorRates[repeatingPatternSize-1][refPatternRepeatCount-1]
reverseIndelType
indelToRefErrorProb = errorRates[repeatingPatternSize-1][indelPatternRepeatCount-1]
```

Somatic Indel Error Model use Non-adaptive indel error model $e_i(1,r) = e_d(1,r) = e_l exp(f_r(log(e_h) - log(e_l))), e_l = 5 * 10^{-5}, e_h = 3 * 10^{-4}$

```
el <- log(5*10e-5)
eh <- log(3*10e-4)
C <- 15
S <- 1
for p <- 1 to C
    highErrorFrac <- min(p-1, C) / C
    lowErrorFrac <- (1.0 - highErrorFrac) * el - highErrorFrac * eh
    errorRate[S-1][p-1] <- exp(logErrorRate)
```

Realign And Score Read

- 1 Indel Candidacy **check__for__candidate__indel__overlap** check candidate indel by implement **isCandidateIndelImpl** Given a total locus coverage of N, Indel coverage of n_i, and an indel error of e_l, we define the probability of some coverage x beging generaterd at the locus due to indel error
  $P(x|N, e_l) = C_N^x e_i^x (1 - e_l)^{N-x}$
  $P(X >= n_i|N, e_l) = \sum_{x=n_i}^{N} P(x|N, e_l)$
  $p_{reject} = 1e^{-9}, e_l = 5e^{-5}$

```
Function  isCandidateIndelImplTestSignalNoise
for sample in [normal, turmor]
    N <- totalReadCount
    N <- max (N ,tier1ReadSupportCount)
    p <- 5e-5
    n_success <- tier1ReadSupportCount
    alpha <- 1e-9
```

- 2 Read realignment **Function 2.1** getCandidateAlignments, to initialize a candidate alignment from a standard alignment. **Function 2.2** add__indels__in__range, find all indels in the indel__buffer which intersect a range (and meet candidacy/usability requirements) **Function 2.3** getAlignmentIndels, get the keys of the indels present including mismatch in the candidate alignment, **Function 2.4** candidate__alignment__search, recursively build potential alignment paths and push them into the candidate alignment set each recursive step invokes (up to) 3 paths:

1) is the current state of the active indel
2) is the alternate state of the active indel with the alignment's start position pinned
3) is the alternate state of the active indel with the alignment's end position pinned, **get__end__pin__start__pos** work backwards from end__pos to get start__pos and read__start__pos case 2 and case 3 CandidateAlignment implement by **make__start__pos__alignment**

```
Function make_start_pos_alignment
```

```
case 2:
make_start_pos_alignment(ref_start_pos, read_start_pos,
is_fwd_strand, read_length, curIndel)
case 3:
function get_end_pin_start_pos => ref_start_pos and read_start_pos
```

Function 2.5 scoreCandidateAlignmentsAndIndels, Find the most likely alignment and most likely alignment for each indel state for every indel in indel_status_map

Function 2.6 scoreCandidateAlignments, Score of candidate alignment cal for read segment rseg. P(read | haplotype)

$P(d|h) = \prod_k (q_k if(d_k = a_k)|(1 - q_k)/3 if(d_k \neq a_K)|1/4)$

```
function scoreCandidateAlignments
for cal <- cals
    lnp <- 0.0
    function scoreCandidateAlignment (cal)
        cigar <- cagars
        case cigar.type == Match:
            for i <- i ... cigar.length
            if is_same
                lnp += ln_comp(qscore)
            else
                lnp += ln(qsore) + ln(1/3)
        case cigar.type == Insert:
            IndelKey <- getMatchingIndelKey
            for i <- i .. cigar.length
            read_base <- read[read_pos + i]
            ref_base <- ref[ref_pos]
            if read_base == ref_base
                lnp += ln_comp(qscore)
            else
                lnp += ln(qsore) + ln(1/3)
        case cigar.type == Delete
            IndelKey <- getMatchingIndelKey
        if (IndelKey != INDEL::NONE) && not isCandidate
            lnp += log(1e-5)
```

Function 2.7 score_indels, use the most likely alignment for each indel state for every indel in indel_status_map to generate data needed in indel calling. to compute ReadPathScores

iks_map_t

```
Funtion score_indels
iks_map_t =  map<indel_status_t,align_info_t>
iks_map_t indelScoringInfo;
if isIndelInCandAlignment
// indel present:
    indel_lk = lnp
// indel absent w/o interference:
    ref_lk = lnp + log(5e-5)
//refToIndelErrorProb = log(5e-5)
else
    indel_lk = lnp + log(5e-5)
// indelToRefErrorProb = log(5e-5)
    ref_lk = lnp
```

Funtion 2.8 pileup_pos_reads, Add reads buffered at position into a basecall pileup to allow for downstream depth and site genotyping calculations mapping error adjustment $e_b = (1 - e_m)e_b + e_m 3/4$ compute base_call and insert insert_pos_basecall

Function 2.9 create_mismatch_filter_map

Somatic call variant

- 1 Snp **Function 3.1** process_pos_snp_somatic, actually implement by **position_somatic_snv_call get_diploid_gt_lhood_cached_simple**, compute likelihood of REF, HET, HOM q is qscore of read $ error_prob = 10 ^(-q/10), lnce = log(1 - error_prob), lne = -q/10-log10$
  $P_{ref} = \prod_i pref_i$
  $P_{het} = \prod_i phet_i$
  $P_{hom} = \prod_i phom_i$
  $pref_i = ceprob if a_k = b_k, pref_i = eprob/3 if a_k \neq b_k$
  $phet_i = 1/2(ceprob + eprob/3)$
  $phom_i = ceprob if if a_k \neq b_k, phom_i = eprob/3 if a_k = b_k$

```
get_diploid_gt_lhood_cached_simple
lhood[REF] <- 0.0
lhood[HET] <- 0.0
lhood[HOM] <- 0.0
for bc <- bcs
    eprob <- bc.error_prob
    ceprob <- 1.0 - eprob
    lne <- bc.ln_error_prob
    lnce <- bc.ln_comp_error_prob
    cv[0] <- lne + ln(1/3)
    cv[1] <- ln(ceprob + 1/3 * eprob) + ln(1/2)
    cv[2] <- lnce
    if bc.base == ref.base
        lhood[REF] += cv[2]
        lhood[HET] += cv[1]
        lhood[HOM] += cv[0]
    else
        lhood[REF] += cv[0]
        lhood[HET] += cv[1]
        lhood[HOM] += cv[2]
```

get_diploid_het_grid_lhood_cached, get likelihood of non-canonical frequencies (0.05, 0.1, ..., 0.45, 0.55, ..., 0.95) If read base equal ref base, then
$p_{high} = f_r * ceprob + (1.0 - f_r) * eprob * 1/3)$
$P_{low} = f_r * eprob * 1/3 + (1.0 - f_r) * ceprob)$
else
$p_{low} = f_r * ceprob + (1.0 - f_r) * eprob * 1/3)$
$p_{high} = f_r * eprob * 1/3 + (1.0 - f_r) * ceprob)$

$P_{low} = \prod_i p_{low}, P_{high} = \prod_i p_{high}$

```
get_diploid_het_grid_lhood_cached
for gt = 0; gt<18; ++gt
    lhood[gt] = 0.;
for  het_inde = 0; het_inde < 9; ++het_index
    het_ratio = (het_inde + 1) * 0.05
    chet_ratio <- 1.0 - het_ratio
    for i = 0; i < n_call; ++n
        eprob <- be.error_prob
        ceprob <- 1.0 - eprob
        cv[0] <- ln(ceprob * het_ratio + eprob * 1/3 * chet_ratio)
        cv[1] <- ln(ceprob * chet_ratio + eprob * 1/3* het_ratio)
        if bc.base == ref.base
            lhood_high += cv[0]
            lhood_low += cv[1]
        else
            lhood_high += cv[0]
            lhood_low += cv[1]
```

get_diploid_strand_grid_lhood_spi, get likelihood of strand states (0.05, ..., 0.45)

```
get_diploid_strand_grid_lhood_spi
for (unsigned i(0); i<9; ++i)
    het_ratio <- 0.05 * (i+1)
    for bc <- base_calls
        eprob <- be.error_prob
        ceprob <- 1.0 - eprob
        cv[0] <- ln(ceprob * het_ratio + eprob * 1/3 * chet_ratio)
        cv[1] <- ln(ceprob * chet_ratio + eprob * 1/3* het_ratio)
        if bc.base == ref.base
            val_off_strand <- bc.ln_comp_error_prob()
            val_fwd <- bc.is_fwd_strand? cv.val[0] : val_off_strand
            val_rev <- bc.is_fwd_strand? val_off_strand : cv.val[0])
            lhood_fwd += val_fwd;
            lhood_rev += val_rev;
        else
            val_off_strand <- bc.ln_error_prob()+ln(1/3)
            val_off_strand <- bc.ln_comp_error_prob()
            val_fwd <- bc.is_fwd_strand? cv.val[1] : val_off_strand
            val_rev <- bc.is_fwd_strand? val_off_strand : cv.val[1])
            lhood_fwd += val_fwd;
            lhood_rev += val_rev;
*lhood = getLogSum(lhood_fwd,lhood_rev)+ln_one_half;
```

calculate__result__set__grid, genomic site results:

$P*(G_t,G_n|D) \propto P(Gt,Gn)P(D|G_t,G_n)$
$P(D|G_t,G_n) = \int_{F_t,F_n} P(D|F_t,F_n)P(F_t,F_n|G_t,G_n)$
$P(D|F_t,F_n) = P(D_t|F_t)P(D_n|F_n) = \prod_i (F_t P(D_t j|H_1)+(1.0-F_t)P(D_j|H_2)) \prod_i (F_n P(D_n j|H_1)+(1.0-F_n)P(D_n j|H_2))$

$$P(G_n) = \begin{cases} \theta if G_n = g_{het} \\ \theta/2 if G_n = g_{hom} \\ 1-3\theta/2 if G_n = g_{ref} \end{cases}$$

$$P(Gt,Gn) = \begin{cases} (1-\gamma)P(G_n) & \text{if } G_t = \texttt{nonsom} \\ \gamma P(G_n) & \text{if } G_t = \texttt{som} \end{cases}$$

$\gamma = e^{-4} forsnvs, = e^{-6} forindels$

$$P(F_t,F_n|G_t = nonsom, G_n) = \begin{cases} 0 & \text{if } F_t \neq F_n \\ 1-\mu & \text{if } F_t = F_n and C(F_n,G_n) = 1 \\ \mu U(F_t) & \text{if } F_t = F_n and C(F_n,G_n) = 0 \end{cases}$$

$$P(F_t,F_n|G_t = som, G_n = ref) = \begin{cases} U(F_t)U(F_n|F_t) & \text{if } F_t \neq F_n, F_n \leq \tau F_t and F_n \leq \delta \\ 0 & \text{otherwisw} \end{cases}$$

$$P(F_t,F_n|G_t = som, G_n \neq ref) = \begin{cases} U(F_t) & \text{if } F_t \neq F_n and C(F_n,G_n) = 1 \\ 0 & \text{otherwisw} \end{cases}$$

$\tau = 0.15, \delta = 0.05, \mu_{snv} = 5e^{-10}, \mu_{indel} = e^{2.2}_{ref}$

```
Function calculate_result_set_grid
_contam_tolerance <- 0.15
_ln_sse_rate <- ln(0.0000000005)
_ln_csse_rate <-ln(1.0 -  shared_error_rate)
for ngt = 0; ngt < 3; ++ngt // ngt = ref | het | hom
    for tgt = 0; tgt < 2; ++tgt // non-somatic, 1: somatic
        index = 0
        for tumor_freq_index in [0...21] {
            // 0 -> 0.0 : 1 -> 1.0f : 2 -> 0.5 : other
            tumor_freq <- 0.05 * tumor_freq_index
            for normal_freq_index <- 0...21 {
                if tgt == 0
```

```
                    lprior_freq <- (normal_freq_index == ngt)  ?_ln_csse_rate : _ln_sse_rat
                else
                    if (normal_freq_index != tumor_freq_index)
                        if (ngt != 0)
                            lprior_freq = log_error_mod
                        else
                            consider_norm_contam <- 0.15*tumor_freq >= 0.05
                            if consider_norm_contam
                                if (normal_freq_index == 0)
                                    lprior_freq = log_error_mod
                                else
                                    if ((normal_freq_index == ngt) || (normal_freq_index ==
                                        lprior_freq = log_error_mod + ln_one_half
                                    else
                                        continue
                            else
                                continue
                    else
                        continue
                double lsum = lprior_freq + normal_lhood[normal_freq_index] + tumor_lhood[tu
                log_sum[index++] = lsum;
            }
        }
        double sum = 0.0;
        for (int i(0); i<index; ++i) {}
            sum += std::exp(log_sum[i] - max_log_sum);
        }
        // logP(Gn=ngt, Gt=tgt)
        log_genotype_prior = germlineGenotypeLogPrior[ngt] + ((tgt == 0) ? lnmatch : lnmisma
        // log(P(G)) + log(D|G)
        log_post_prob[ngt][tgt] = log_genotype_prior + max_log_sum + std::log(sum);
        if (log_post_prob[ngt][tgt] > max_log_prob)  {
                max_log_prob = log_post_prob[ngt][tgt];
                rs.max_gt = DDIGT::get_state(ngt, tgt);
        }
// Calculate posterior probabilities ( P(G)*P(D|G) )
 sum_prob = 0.0;
// REF HOM HET
for (ngt(0); ngt<3; ++ngt){
    for (unsigned tgt(0); tgt<2; ++tgt) {
         prob = std::exp(log_post_prob[ngt][tgt] - max_log_prob); // to prevent underflow
        sum_prob += prob;
    }
}
```

```
log_sum_prob = std::log(sum_prob);
min_not_somfrom_sum(INFINITY);
nonsom_prob = 0.0;
double post_prob[3][2];
for (unsigned ngt(0); ngt < 3; ++ngt) {
    som_prob_given_ngt(0);
    for (unsigned tgt(0); tgt<2; ++tgt)    {
        post_prob[ngt][tgt] = std::exp(log_post_prob[ngt][tgt] - max_log_prob - log_sum_prob
        if (tgt == 0) {// Non-somatic
            nonsom_prob += post_prob[ngt][tgt];
        } else {// Somatic
            som_prob_given_ngt += post_prob[ngt][tgt];
        }
    }

    err_som_and_ngt = 1.0 - som_prob_given_ngt;
    if (err_som_and_ngt < min_not_somfrom_sum) {
        min_not_somfrom_sum=err_som_and_ngt;
        rs.from_ntype_qphred=error_prob_to_qphred(err_som_and_ngt);
        rs.ntype=ngt;
    }
}
rs.qphred=error_prob_to_qphred(nonsom_prob);
```

- 2 Indel **Function 3.2** process_pos_indel_somatic, **Function 3.3** get_indel_digt_lhood **integrateOutMappingStatus**, Given the log-likelihood of the read conditioned on correct mapping as input, sum over the correct and incorrect mapping states to approximately integrate out the mapping status condition.

  ```
  integrateOutMappingStatus
  correctMappingLogPrior=std::log(1.7e-10);
  dopt.randomBaseMatchLogProb = log(0.5)
  dopt.tier2RandomBaseMatchLogProb = log(0.25)
  thisRandomBaseMatchLogLikelihood = isTier2 ? log(0.25) : log(0.5)
  a = correctMappingLogLikelihood + correctMappingLogPrior
  b = read_length * thisRandomBaseMatchLogLikelihood
  getLogSum (a, b)
  ```

  get_het_observed_allele_ratio get expected allele ratio between two indel alleles (where one allele must be reference)

  ```
  get_het_observed_allele_ratio(    const unsigned read_length,
  const unsigned min_overlap,
  const IndelKey& indelKey,
  const double het_allele_ratio,
  double& log_ref_prob,
  double& log_indel_prob)
  ```

```
     base_expect = read_length+1 - 2 *min_overlap
     ref_path_expect = base_expect + min(indelKey.delete_length, base_expect)
     indel_path_expect = base_expect + min(indelKey.insert_length,base_expect)
     ref_path_term = (1.0 - het_allele_ratio) * ref_path_expect
     indel_path_term =  het_allele_ratio * indel_path_expect
     total_path_term = ref_path_term+indel_path_term
     indel_prob(indel_path_term/total_path_term);
     log_ref_prob=std::log(1.-indel_prob);
     log_indel_prob=std::log(indel_prob)

Function get_indel_digt_lhood
// function score_indel compute ReadPathScores
noindel_lnp = path_lnp.ref
hom_lnp = path_lnp.indel
for indeldata <- indelSampleData
    noindel_lnp <- indeldata.ref
    hom_lnp <- indelData.indel
    log_ref_prob <- ln(1/2)
    log_indel_prob <- ln(1/2)
    het_allele_ratio <- 0.5
    ref_path_term <- 1.0 - het_allele_ratio * ref_path_expect
    indel_path_term <- het_allele_ratio * indel_path_expect
    total_path_term <- ref_path_term+indel_path_term
    indel_prob(indel_path_term/total_path_term);
    log_ref_prob=std::log(1.-indel_prob);
    log_indel_prob=std::log(indel_prob)
    het_lnp = getLogSum(noindel_lnp+log_ref_prob,hom_lnp+log_indel_prob)
    lhood[noIndel] += integrateOutMappingStatus(noindel_lnp)
    lhood[HOM] +=  integrateOutMappingStatus(hom_lnp)
    lhood[Ref] +=  integrateOutMappingStatus(het_lnp)

get_indel_het_grid_lhood
for i(0); i< 9 ; ++i)
    het_ratio = (i + 1) * 0.05
    chet_ratio(1.-het_ratio);
    log_het_ratio= std::log(het_ratio)
    log_chet_ratio = std::log(chet_ratio)

    {
        log_ref_prob = log_chet_ratio);
        log_indel_prob = log_het_ratio);
        if (! is_breakpoint) {
            get_het_observed_allele_ratio(path_lnp.read_length,sample_opt.min_read_bp_flank
        }
        het_lnp = (getLogSum(noindel_lnp+log_ref_prob, hom_lnp+log_indel_prob));
        het_lhood_low += integrateOutMappingStatus(dopt, path_lnp.nonAmbiguousBasesInRead,
    }
    {
        double log_ref_prob(log_het_ratio);
        double log_indel_prob(log_chet_ratio);
        if (! is_breakpoint)  {
            get_het_observed_allele_ratio(path_lnp.read_length,sample_opt.min_read_bp_flank,
        }
        const double het_lnp(getLogSum(noindel_lnp+log_ref_prob, hom_lnp+log_indel_prob));
        het_lhood_high += integrateOutMappingStatus(dopt, path_lnp.nonAmbiguousBasesInRead,
}
```

$sharedIndelErrorRate = e_{ref}^{2.2}, e_{ref} = 5e^{-5}$
logSharedIndelErrorRate = log(sharedIndelErrorRate) , logSharedIndelErrorRateComplement = log1p(-logSharedIndelErrorRate) **calculate_result_set_grid** the same to SNV calculate_result_set_grid function