Model Checking Games and a Genome Sequence Search

Sergey Staroletov

Barnaul, February 19, 2021

The paper

S M Staroletov 2020. Model checking games and a genome sequence search. J. Phys.: Conf. Ser. 1679 032020

Introduction

- To foster interest in logical methods, in particular, formal verification, it is advisable to train such methods using games and puzzles
- However, the same methods can be used to solve real problems
- In this paper, we show the use of non-deterministic programming for the task of finding a pattern in a genome sequence

Motivation

- Describe a methodology to solve algorithmic puzzles by the negation of an LTL formula
- Make first steps into computational biology
- Touch “Swarm model checking”
- Cite friends
- Scopus++

Reviewers' comments

- Devoted to model checking, accept
- Non serious, reject
- Too much code
- Issues in RNA and DNA description (resolved)

In this paper, we

1. focus on the model checking games concept
2. show how to encode real algorithms in Promela
3. discuss an effective string comparison implementation in Promela
4. move to fuzzy string comparison
5. apply it in the genome sequence search
6. discuss ways how to solve hard computation tasks in model checking using swarm model checking and state hashing approaches.

Preliminaries

Preliminaries

- Model checking with SPIN
- Bitstate hashing and hashcompact
- Swarm model checking
- SARS-CoV-2 genome: related information
- Substring search methods

Model checking with SPIN

- SPIN[1] is a utility for model checking the correctness of distributed software. The abbreviation SPIN stands for Simple Promela INterpreter
- The SPIN system verifies not the programs themselves, but their models
- To build a model for an original parallel program or an algorithm, the verifying engineer (usually manually) builds a representation of this program in a C-like input language, called Promela (Protocol MEta-LAnguage)

[1] Holzmann G J 1997 The model checker SPIN

Model checking with SPIN

In this paper, we rely on the following language features:

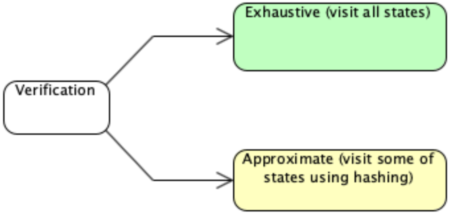
- the presence of arrays;
- the presence of *do-while* loops;
- the presence of *if* clause including the non-deterministic choice.

As well as we use the following SPIN model checker features:

- checking of LTL properties expressed in predicates with key program variables;
- ability to present a counter-example as a trail of visited states if the LTL property does not hold;
- optimized depth-first search (DFS);
- bitstate hashing to dramatically reduce used memory;
- ability to parallelize the model checking process using the swarm technique.

Bitstate hashing and hashcompact

In order to reduce memory for storing the states, in addition to strict (*exhaustive*) verification, SPIN offers hashing methods to do the checking that can visit most of the states until a hash collision has not occurred.


Verification
Exhaustive (visit all states)
Approximate (visit some of states using hashing)

Bitstate hashing and hashcompact

- In such case, for every state of S bits, a hash value of m bits is computed, which is associated with a m unique bit position within a large bit array of size 2^m [2].
- For every new hash value generated the tool inspects the current value of the bit that corresponds to the hash value, and if it is zero, set it to one. If the bit is already set, it counts this as a hash collision.
- Supplementary, the SPIN tool by default uses two hash functions, and stores two bits in the bit array for every state. A hash collision now requires a collision on both bits.

[2]Holzmann G J 1998 An analysis of bitstate hashing

Bitstate hashing and hashcompact

An alternative strategy recommended by Wolper[3] is called hashcompact. In the hashcompact method, the state descriptor is compressed from S bits to 64 bits, using a single hash function. The resulting 64-bit values are then stored in a normal lookup table with collision resolution.

[3] Wolper P and Leroy D 1993 Reliable hashing without collision detection

Swarm model checking

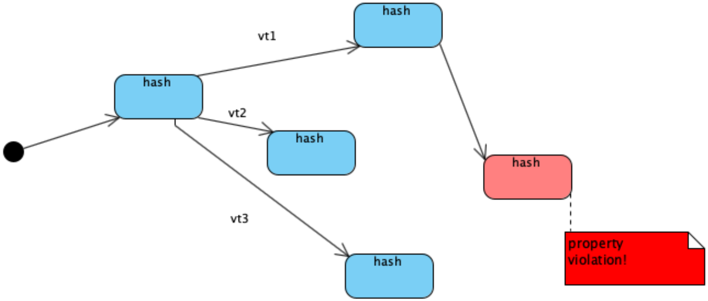
Swarm model checking is an approach to generate and run a bunch of verification tests (VTs) by combining three basic ideas to modify the search process[4]:

- search randomization (use different seed values for non-deterministic choices);
- search diversification (performing searches forwards or in reverse, varying hashing options);
- search parallelization (run multiple VTs in parallel).

Swarm is implemented using a pre-processor tool that generates a script to compile different VTs from an input Promela model and runs them.

[4]Holzmann G J, Joshi R and Groce A 2008 Swarm verification

Swarm model checking



SARS-CoV-2 genome: related information

SARS-CoV-2, the coronavirus that cases CoVID-19 pandemic, is having a strong influence to the world economy, led to thousands of deads and changed plans of billions of people; in the other side, it catalyzes the processes of digitalization and puts an enormous interest to research in the sphere of biology and computational biology.

SARS-CoV-2 genome: related information

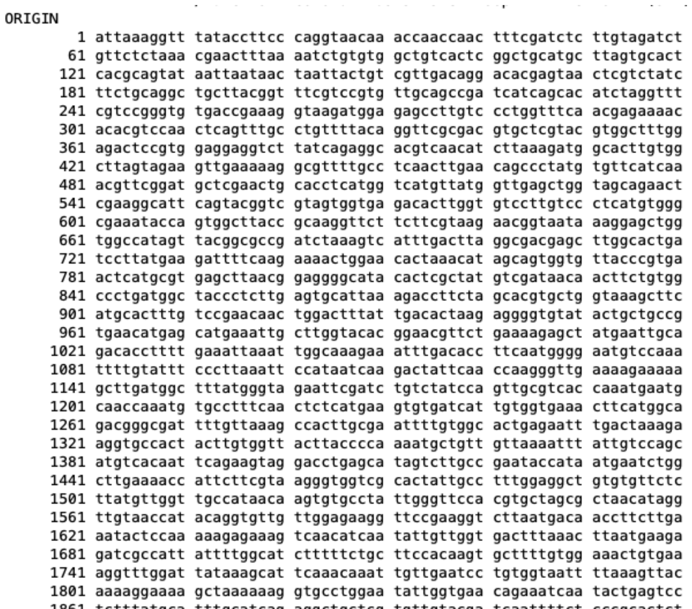
- The coronavirus genome has been already decoded and is available in[5]
- Wu et al.[6] analyzed the genome and found that it is 89% similar to the bat coronavirus bat-SL-CoVZC45[7]
- The viral genome is represented as a single-stranded RNA, which consists of adenine (A), guanine (G), cytosine (C) and uracil (U)
- The uracil symbol is often represented as thymine (T) in the sequence to do proper software support
- So we have a string of 29903 nucleotides with alphabet {A,G,C,T}.

[5]Severe acute respiratory syndrome coronavirus 2 isolate Wuhan-Hu-1, complete genome URL: https://www.ncbi.nlm.nih.gov/nuccore/NC045512

[6]Wu F, Zhao S, Yu B, Chen Y M, Wang W, Song Z G, Hu Y, Tao Z W, Tian J H, Pei Y Yet al. 2020 A new coronavirus associated with human respiratory disease in China

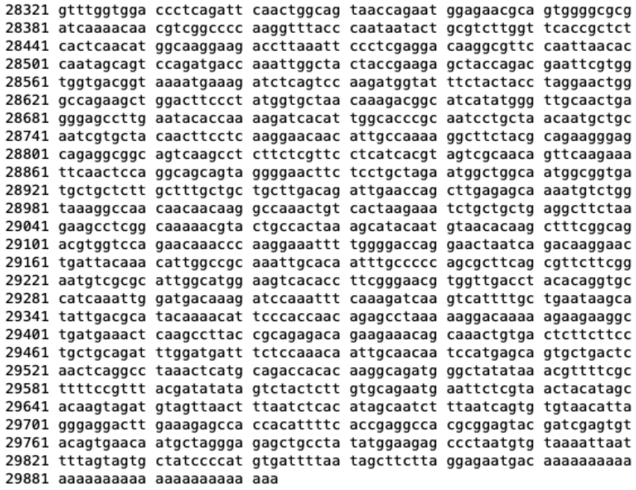
[7]Bat SARS-like coronavirus isolate bat-SL-CoVZC45, complete genome URL:https://www.ncbi.nlm.nih.gov/nuccore/MG772933

SARS-CoV-2 genome: related information



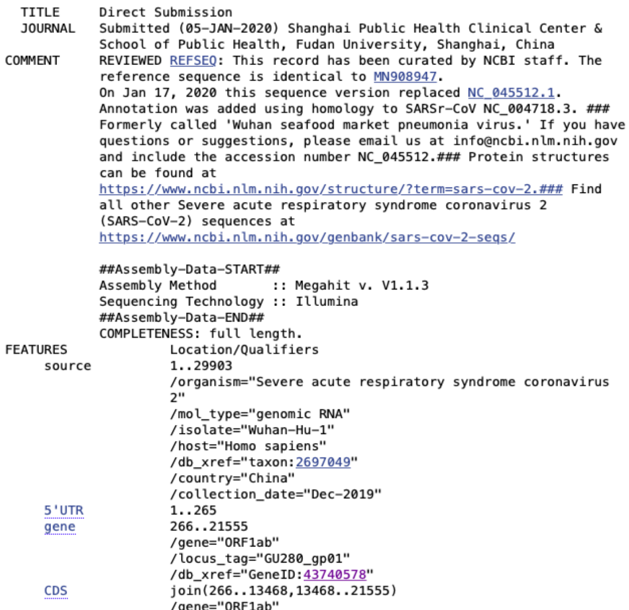
```
ORIGIN
        1 attaaaggtt tataccttcc caggtaacaa accaaccaac tttcgatctc ttgtagatct
       61 gttctctaaa cgaactttaa aatctgtgtg gctgtcactc ggctgcatgc ttagtgcact
      121 cacgcagtat aattaataac taattactgt cgttgacagg acacgagtaa ctcgtctatc
      181 ttctgcaggc tgcttacggt ttcgtccgtg ttgcagccga tcatcagcac atctaggttt
      241 cgtccgggtg tgaccgaaag gtaagatgga gagccttgtc cctggtttca acgagaaaac
      301 acacgtccaa ctcagtttgc ctgttttaca ggttcgcgac gtgctcgtac gtggctttgg
      361 agactccgtg gaggaggtct tatcagaggc acgtcaacat cttaaagatg gcacttgtgg
      421 cttagtagaa gttgaaaaag gcgttttgcc tcaacttgaa cagccctatg tgttcatcaa
      481 acgttcggat gctcgaactg cacctcatgg tcatgttatg gttgagctgg tagcagaact
      541 cgaaggcatt cagtacggtc gtagtggtga gacacttggt gtccttgtcc ctcatgtggg
      601 cgaaatacca gtggcttacc gcaaggttct tcttcgtaag aacggtaata aaggagctgg
      661 tggccatagt tacggcgccg atctaaagtc atttgactta ggcgacgagc ttggcactga
      721 tccttatgaa gattttcaag aaaactggaa cactaaacat agcagtggtg ttacccgtga
      781 actcatgcgt gagcttaacg gaggggcata cactcgctat gtcgataaca acttctgtgg
      841 ccctgatggc taccctcttg agtgcattaa agaccttcta gcacgtgctg gtaaagcttc
      901 atgcactttg tccgaacaac tggactttat tgacactaag aggggtgtat actgctgccg
      961 tgaacatgag catgaaattg cttggtacac ggaacgttct gaaaagagct atgaattgca
     1021 gacacctttt gaaattaaat tggcaaagaa atttgacacc ttcaatgggg aatgtccaaa
     1081 ttttgtattt cccttaaatt ccataatcaa gactattcaa ccaagggttg aaaagaaaaa
     1141 gcttgatggc tttatgggta gaattcgatc tgtctatcca gttgcgtcac caaatgaatg
     1201 caaccaaatg tgcctttcaa ctctcatgaa gtgtgatcat tgtggtgaaa cttcatggca
     1261 gacgggcgat tttgttaaag ccacttgcga attttgtggc actgagaatt tgactaaaga
     1321 aggtgccact acttgtggtt acttacccca aaatgctgtt gttaaaattt attgtccagc
     1381 atgtcacaat tcagaagtag gacctgagca tagtcttgcc gaataccata atgaatctgg
     1441 cttgaaaacc attcttcgta agggtggtcg cactattgcc tttggaggct gtgtgttctc
     1501 ttatgttggt tgccataaca agtgtgccta ttgggttcca cgtgctagcg ctaacatagg
     1561 ttgtaaccat acaggtgttg ttggagaagg ttccgaaggt cttaatgaca accttcttga
     1621 aatactccaa aaagagaaag tcaacatcaa tattgttggt gactttaaac ttaatgaaga
     1681 gatcgccatt attttggcat ctttttctgc ttccacaagt gcttttgtgg aaactgtgaa
     1741 aggtttggat tataaagcat tcaaacaaat tgttgaatcc tgtggtaatt ttaaagttac
     1801 aaaaggaaaa gctaaaaaag gtgcctggaa tattggtgaa cagaaatcaa tactgagtcc
```

SARS-CoV-2 genome: related information



```
28321 gtttggtgga ccctcagatt caactggcag taaccagaat ggagaacgca gtggggcgcg
28381 atcaaaacaa cgtcggcccc aaggtttacc caataatact gcgtcttggt tcaccgctct
28441 cactcaacat ggcaaggaag accttaaatt ccctcgagga caaggcgttc caattaacac
28501 caatagcagt ccagatgacc aaattggcta ctaccgaaga gctaccagac gaattcgtgg
28561 tggtgacggt aaaatgaaag atctcagtcc aagatggtat ttctactacc taggaactgg
28621 gccagaagct ggacttccct atggtgctaa caaagacggc atcatatggg ttgcaactga
28681 gggagccttg aatacaccaa aagatcacat tggcacccgc aatcctgcta acaatgctgc
28741 aatcgtgcta caacttcctc aaggaacaac attgccaaaa ggcttctacg cagaagggag
28801 cagaggcggc agtcaagcct cttctcgttc ctcatcacgt agtcgcaaca gttcaagaaa
28861 ttcaactcca ggcagcagta ggggaacttc tcctgctaga atggctggca atggcggtga
28921 tgctgctctt gctttgctgc tgcttgacag attgaaccag cttgagagca aaatgtctgg
28981 taaaggccaa caacaacaag gccaaactgt cactaagaaa tctgctgctg aggcttctaa
29041 gaagcctcgg caaaaacgta ctgccactaa agcatacaat gtaacacaag ctttcggcag
29101 acgtggtcca gaacaaaccc aaggaaattt tggggaccag gaactaatca gacaaggaac
29161 tgattacaaa cattggccgc aaattgcaca atttgccccc agcgcttcag cgttcttcgg
29221 aatgtcgcgc attggcatgg aagtcacacc ttcgggaacg tggttgacct acacaggtgc
29281 catcaaattg gatgacaaag atccaaattt caaagatcaa gtcattttgc tgaataagca
29341 tattgacgca tacaaaacat tcccaccaac agagcctaaa aaggacaaaa agaagaaggc
29401 tgatgaaact caagccttac cgcagagaca gaagaaacag caaactgtga ctcttcttcc
29461 tgctgcagat ttggatgatt tctccaaaca attgcaacaa tccatgagca gtgctgactc
29521 aactcaggcc taaactcatg cagaccacac aaggcagatg ggctatataa acgttttcgc
29581 ttttccgttt acgatatata gtctactctt gtgcagaatg aattctcgta actacatagc
29641 acaagtagat gtagttaact ttaatctcac atagcaatct ttaatcagtg tgtaacatta
29701 gggaggactt gaaagagcca ccacattttc accgaggcca cgcggagtac gatcgagtgt
29761 acagtgaaca atgctaggga gagctgccta tatggaagag ccctaatgtg taaaattaat
29821 tttagtagtg ctatccccat gtgattttaa tagcttctta ggagaatgac aaaaaaaaaa
29881 aaaaaaaaaa aaaaaaaaaa aaa
```

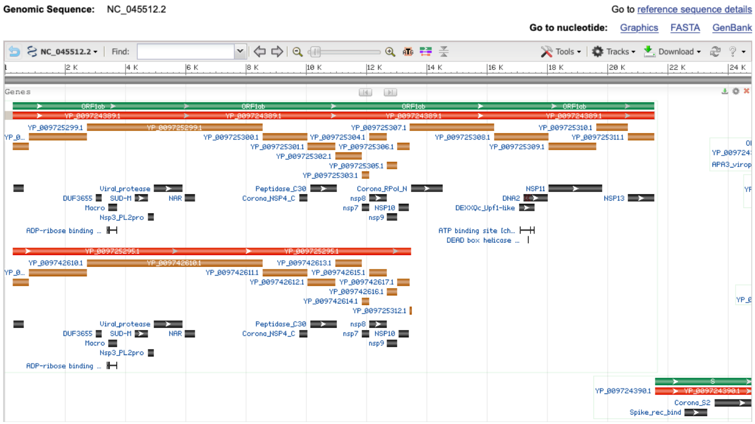
SARS-CoV-2 genome: related information



```
  TITLE     Direct Submission
  JOURNAL   Submitted (05-JAN-2020) Shanghai Public Health Clinical Center &
            School of Public Health, Fudan University, Shanghai, China
COMMENT     REVIEWED REFSEQ: This record has been curated by NCBI staff. The
            reference sequence is identical to MN908947.
            On Jan 17, 2020 this sequence version replaced NC_045512.1.
            Annotation was added using homology to SARSr-CoV NC_004718.3. ###
            Formerly called 'Wuhan seafood market pneumonia virus.' If you have
            questions or suggestions, please email us at info@ncbi.nlm.nih.gov
            and include the accession number NC_045512.### Protein structures
            can be found at
            https://www.ncbi.nlm.nih.gov/structure/?term=sars-cov-2.### Find
            all other Severe acute respiratory syndrome coronavirus 2
            (SARS-CoV-2) sequences at
            https://www.ncbi.nlm.nih.gov/genbank/sars-cov-2-seqs/

            ##Assembly-Data-START##
            Assembly Method       :: Megahit v. V1.1.3
            Sequencing Technology :: Illumina
            ##Assembly-Data-END##
            COMPLETENESS: full length.
FEATURES             Location/Qualifiers
     source          1..29903
                     /organism="Severe acute respiratory syndrome coronavirus
                     2"
                     /mol_type="genomic RNA"
                     /isolate="Wuhan-Hu-1"
                     /host="Homo sapiens"
                     /db_xref="taxon:2697049"
                     /country="China"
                     /collection_date="Dec-2019"
     5'UTR           1..265
     gene            266..21555
                     /gene="ORF1ab"
                     /locus_tag="GU280_gp01"
                     /db_xref="GeneID:43740578"
     CDS             join(266..13468,13468..21555)
                     /gene="ORF1ab"
```

SARS-CoV-2 genome: related information



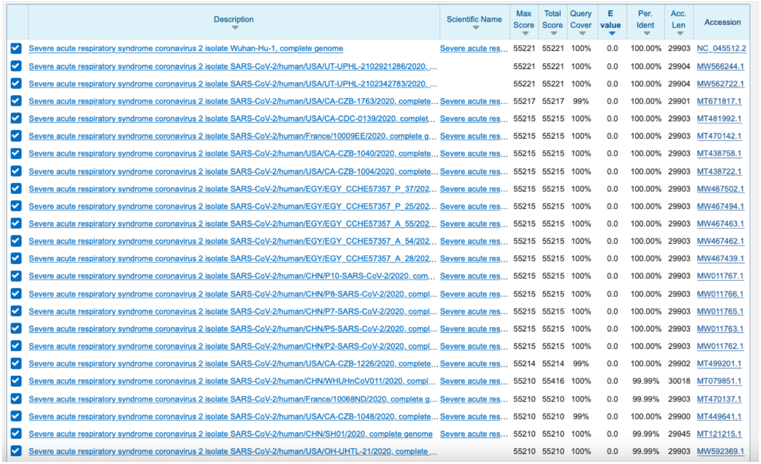
Genomic Sequence: NC_045512.2

Go to reference sequence details

Go to nucleotide: Graphics FASTA GenBank

SARS-CoV-2 genome: related information

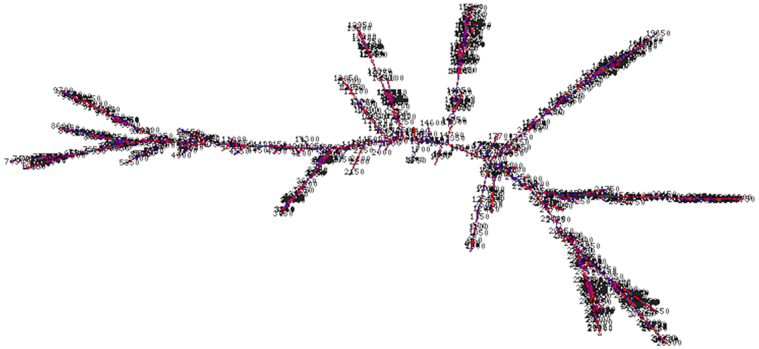
The search of similarities with BLAST:



	Description	Scientific Name	Max Score	Total Score	Query Cover	E value	Per. Ident	Acc. Len	Accession
✓	Severe acute respiratory syndrome coronavirus 2 isolate Wuhan-Hu-1, complete genome	Severe acute res...	55221	55221	100%	0.0	100.00%	29903	NC_045512.2
✓	Severe acute respiratory syndrome coronavirus 2 isolate SARS-CoV-2/human/USA/UT-UPHL-2102921286/2020, ...		55221	55221	100%	0.0	100.00%	29904	MW566244.1
✓	Severe acute respiratory syndrome coronavirus 2 isolate SARS-CoV-2/human/USA/UT-UPHL-2102342783/2020, ...		55221	55221	100%	0.0	100.00%	29904	MW562722.1
✓	Severe acute respiratory syndrome coronavirus 2 isolate SARS-CoV-2/human/USA/CA-CZB-1763/2020, complete...	Severe acute res...	55217	55217	99%	0.0	100.00%	29901	MT671817.1
✓	Severe acute respiratory syndrome coronavirus 2 isolate SARS-CoV-2/human/USA/CA-CDC-0139/2020, complet...	Severe acute res...	55215	55215	100%	0.0	100.00%	29903	MT481992.1
✓	Severe acute respiratory syndrome coronavirus 2 isolate SARS-CoV-2/human/France/10009EE/2020, complete g...	Severe acute res...	55215	55215	100%	0.0	100.00%	29903	MT470142.1
✓	Severe acute respiratory syndrome coronavirus 2 isolate SARS-CoV-2/human/USA/CA-CZB-1040/2020, complete...	Severe acute res...	55215	55215	100%	0.0	100.00%	29903	MT438758.1
✓	Severe acute respiratory syndrome coronavirus 2 isolate SARS-CoV-2/human/USA/CA-CZB-1004/2020, complete...	Severe acute res...	55215	55215	100%	0.0	100.00%	29903	MT438722.1
✓	Severe acute respiratory syndrome coronavirus 2 isolate SARS-CoV-2/human/EGY/EGY_CCHE57357_P_37/202...	Severe acute res...	55215	55215	100%	0.0	100.00%	29903	MW467502.1
✓	Severe acute respiratory syndrome coronavirus 2 isolate SARS-CoV-2/human/EGY/EGY_CCHE57357_P_25/202...	Severe acute res...	55215	55215	100%	0.0	100.00%	29903	MW467494.1
✓	Severe acute respiratory syndrome coronavirus 2 isolate SARS-CoV-2/human/EGY/EGY_CCHE57357_A_55/202...	Severe acute res...	55215	55215	100%	0.0	100.00%	29903	MW467463.1
✓	Severe acute respiratory syndrome coronavirus 2 isolate SARS-CoV-2/human/EGY/EGY_CCHE57357_A_54/202...	Severe acute res...	55215	55215	100%	0.0	100.00%	29903	MW467462.1
✓	Severe acute respiratory syndrome coronavirus 2 isolate SARS-CoV-2/human/EGY/EGY_CCHE57357_A_28/202...	Severe acute res...	55215	55215	100%	0.0	100.00%	29903	MW467439.1
✓	Severe acute respiratory syndrome coronavirus 2 isolate SARS-CoV-2/human/CHN/P10-SARS-CoV-2/2020, com...	Severe acute res...	55215	55215	100%	0.0	100.00%	29903	MW011767.1
✓	Severe acute respiratory syndrome coronavirus 2 isolate SARS-CoV-2/human/CHN/P8-SARS-CoV-2/2020, compl...	Severe acute res...	55215	55215	100%	0.0	100.00%	29903	MW011766.1
✓	Severe acute respiratory syndrome coronavirus 2 isolate SARS-CoV-2/human/CHN/P7-SARS-CoV-2/2020, compl...	Severe acute res...	55215	55215	100%	0.0	100.00%	29903	MW011765.1
✓	Severe acute respiratory syndrome coronavirus 2 isolate SARS-CoV-2/human/CHN/P5-SARS-CoV-2/2020, compl...	Severe acute res...	55215	55215	100%	0.0	100.00%	29903	MW011763.1
✓	Severe acute respiratory syndrome coronavirus 2 isolate SARS-CoV-2/human/CHN/P2-SARS-CoV-2/2020, compl...	Severe acute res...	55215	55215	100%	0.0	100.00%	29903	MW011762.1
✓	Severe acute respiratory syndrome coronavirus 2 isolate SARS-CoV-2/human/USA/CA-CZB-1226/2020, complete...	Severe acute res...	55214	55214	99%	0.0	100.00%	29902	MT499201.1
✓	Severe acute respiratory syndrome coronavirus 2 isolate SARS-CoV-2/human/CHN/WHUHnCoV011/2020, compl...	Severe acute res...	55210	55416	100%	0.0	99.99%	30018	MT079851.1
✓	Severe acute respiratory syndrome coronavirus 2 isolate SARS-CoV-2/human/France/10068ND/2020, complete g...	Severe acute res...	55210	55210	100%	0.0	99.99%	29903	MT470137.1
✓	Severe acute respiratory syndrome coronavirus 2 isolate SARS-CoV-2/human/USA/CA-CZB-1048/2020, complete...	Severe acute res...	55210	55210	99%	0.0	100.00%	29900	MT449641.1
✓	Severe acute respiratory syndrome coronavirus 2 isolate SARS-CoV-2/human/CHN/SH01/2020, complete genome	Severe acute res...	55210	55210	100%	0.0	99.99%	29945	MT121215.1
✓	Severe acute respiratory syndrome coronavirus 2 isolate SARS-CoV-2/human/USA/OH-UHTL-21/2020, complete ...		55210	55210	100%	0.0	99.99%	29903	MW592369.1

SARS-CoV-2 genome: related information

A graphical representation of the viral genome that was built using a patched version of mfold software:



SARS-CoV-2 genome: related information

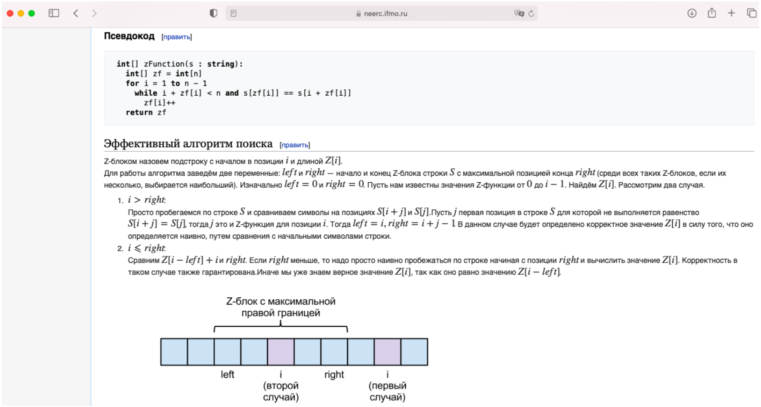
One of the main tasks in this field is the comparison of the genomes, that can help to investigate from whose animals the virus has come and which parts of them are changed due to mutations. The latter brings us to the problem of string comparison, and the comparison should be fuzzy.

Substring search methods

- In a naive algorithm, the search for all admissible shifts is performed using a cycle in which the condition for the equality of the current characters of the string and the pattern is checked. Such an algorithm has $O(N^2)$ complexity, where N corresponds to the length of the string (we will use $|string|$ for the length)
- There are plenty of algorithms to do the searching more effectively (including hashing, trie, suffix automaton, Knuth–Morris–Pratt automaton)
- In this work, we consider the Z-function algorithm which is popular in the competitive programming contests and requires $|string|+|pattern|$ additional memory, and has the complexity of $O(N)$ to build the Z-function and $O(N)$ to search the pattern
- It is fast and requires no pointers or complex data structures so it could be implemented in such a modeling language as Promela

Substring search methods

IFMO wiki:



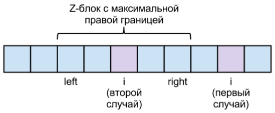
Псевдокод [править]

```
int[] zFunction(s : string):
  int[] zf = int[n]
  for i = 1 to n − 1
    while i + zf[i] < n and s[zf[i]] == s[i + zf[i]]
      zf[i]++
  return zf
```

Эффективный алгоритм поиска [править]

Z-блоком назовем подстроку с началом в позиции i и длиной $Z[i]$.
Для работы алгоритма заведём две переменные: $left$ и $right$ — начало и конец Z-блока строки S с максимальной позицией конца $right$ (среди всех таких Z-блоков, если их несколько, выбирается наибольший). Изначально $left = 0$ и $right = 0$. Пусть нам известны значения Z-функции от 0 до $i - 1$. Найдём $Z[i]$. Рассмотрим два случая.

1. $i > right$:
   Просто пробегаемся по строке S и сравниваем символы на позициях $S[i + j]$ и $S[j]$.Пусть j первая позиция в строке S для которой не выполняется равенство $S[i + j] = S[j]$, тогда j это и Z-функция для позиции i. Тогда $left = i, right = i + j - 1$ В данном случае будет определено корректное значение $Z[i]$ в силу того, что оно определяется наивно, путем сравнения с начальными символами строки.
2. $i \leqslant right$:
   Сравним $Z[i - left] + i$ и $right$. Если $right$ меньше, то надо просто наивно пробежаться по строке начиная с позиции $right$ и вычислить значение $Z[i]$. Корректность в таком случае также гарантирована.Иначе мы уже знаем верное значение $Z[i]$, так как оно равно значению $Z[i - left]$.



Substring search methods

The Z-function from string S is an array Z, each element Z[i] of which is the length of the longest common prefix between S and the suffix of S starting at i:

$$Z[i](s) = max\{k\} : s[i, ..., i+k] = s[0, ..., k] \tag{1}$$

Substring search methods

The pseudocode to build the Z-function in a loop through a given string and a pattern according to (1):

```
int[] zFunction(s : string):
 int[] zf = int[|s|]
 int left = 0, right = 0
 for i = 1 to |s| - 1
  zf[i] = max(0,
  min(right - i, zf[i - left]))
 while i + zf[i] < |s| and
 s[zf[i]] == s[i + zf[i]]
  zf[i]++
  if i + zf[i] > right
   left = i
   right = i + zf[i]
 return zf
```

Substring search methods

The pseudocode to find a substring *pattern* in a string *text* using the Z-function:

```
int patternSearch(text:string,
pattern:string):
 int[] zf = zFunction(pattern + '#' + text)
 for i = |pattern| + 1 to |text| + 1
  if zf[i] == |pattern| return i
```

Substring search methods

Using Promela language features, we prepared the following implementation of the Z-function construction in Promela:

```
short
zf[STR_SIZE+PATTERN_SIZE+1];
inline S(j, ret) {
if
::(j < PATTERN_SIZE) ->
 ret = pattern[j];
::(j == PATTERN_SIZE) ->
 ret = EPS; //#
::(j > PATTERN_SIZE) ->
 ret = text[j - PATTERN_SIZE - 1];
fi
}
```

Substring search methods

```
inline MAX(a, b, ret) {
if
 ::(a >= b) -> ret = a;
 ::else -> ret = b;
fi
}
inline MIN(a, b, ret) {
if
 ::(a < b) -> ret = a;
 ::else -> ret = b;
fi
}
```

Substring search methods

```
//building the Z-function
int left = 0;
int right = 0;
int n = STR_SIZE +
PATTERN_SIZE + 1;
int i = 1;
do
 ::(pos1 < MAX1) -> {
 printf("%d␣\n", i);
 int min = 0;
 MIN((right - i),
 zf[i - left], min);
 int max = 0;
 MAX(0, min, max);
 zf[i] = max;
 bool isOk = true;
```

Substring search methods

```
do
::isOk -> {
 short s1 = 0;
 S(zf[i], s1);
 short s2 = 0;
S((i + zf[i]), s2);
 isOk = (i + zf[i] < n)
 && (s1 == s2);
 if
  ::isOk -> zf[i] = zf[i] + 1;
  ::else -> skip;
 fi
}
::else -> break;
od
```

Substring search methods

```
 if
  ::((i + zf[i]) > right) -> {
  left = i;
  right = i + zf[i];
 }
  ::else -> skip;
  fi
  i = i + 1;
 }
 ::else -> break;
od
```

Substring search methods

It is pretty similar to the algorithmic pseudocode that we have shown before, but this implementation opens some doors to use the formal verification and model checking games for fuzzy comparing of genomes.

Model checking games

Model checking games

- The concept of model checking games originated from logic and theoretical model checking
- The evaluation of logical formulae can be described by such games, played by two players on an arena which is formed as the product of a structure *K* and a formula ψ
- One player (Verifier) attempts to prove that ψ is satisfied in *K* while the other (Falsifier) tries to refute this[8]
- Earlier, this formalism was used in[9] to play property checking games in a process calculus and modal μ logic with pre-defined rules for players' moves. This formalism is used to study a particular logic and construct wining strategies

[8] Fischer D, Gradel E and Kaiser 2010 Model checking games for the quantitative μ-calculus

[9] Stevens P and Stirling C 1998 Practical model-checking using games

Model checking games

- Recently presented[10] Differential Hybrid games are contests of two players, called Angel and Demon, over hybrid program α and property ϕ that is $[\alpha]\phi$ and $< \alpha > \neg\phi$ refer to complementary winning conditions (ϕ for Demon, $\neg\phi$ for Angel)
- The achievements in this theory can be used to construct cooperative hybrid systems

[10] Platzer A 2017 Differential hybrid games

Model checking games

In this work, we proceed to a different way: the model checking game will have two players (a user and a model checker), the user declares that the system does not satisfy formula ϕ and the model checker tries to refute it and provide a counter-example.

Model checking games

In Karpov's book[11] the method of puzzle solving by the model checker was introduced by the example of *wolf, goat and cabbage problem*. The idea of the puzzle is as follows:

> *It is necessary to transfer the both three alive to the different side of a river using series of trips in a boat that only carry two objects and a ferryman. While the heroes stay steady in the presence of the man, but there exist some restrictions while they stay alone on one and the other side of the river: the wolf can eat the goat and the goat can eat the cabbage.*

A domain-specific approach to construct and solve the task is given in[12].

[11]Karpov Y G 2010 Model checking. Verification of parallel and distributed program systems (in Russian) ISBN 978-5-9775-0404-1

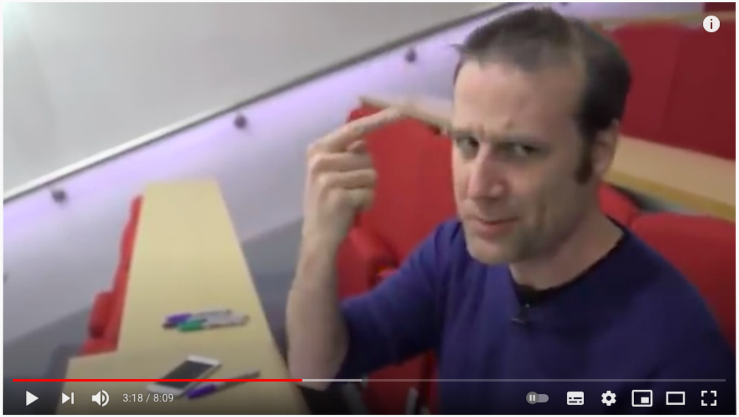
[12]Baar T 2015 A DSL and a SPIN-frontend for river-crossing problems defined with Xtext

Model checking games

Model checking games

The task can be solved using a recursive DFS algorithm, by trying a path of transfers for different objects with these restrictions. Using model checking, it is proposed to encode the state of the system and the rules of changing the state, then create an LTL rule that *”Always the finite state will not be reached”* and if the solution really exists, the model checker can find a path to the finite state and present it as a counterexample. And the state trail to the end state becomes the solution of the problem.

Model checking games



Проблема числа 10958 [Numberphile]

Model checking games

We describe the approach using another different simple *Numeric puzzle* that requires not so much coding.
Let there be a number n.

- If it is even, divide it by 2, i.e. $n \Rightarrow n/2$
- If it is odd, multiply by 3 and add 1, i.e. $n \Rightarrow 3n + 1$
- And repeat the actions until n achieves 1.

  If we start with the number 7, is it possible to get 1?

Model checking games

To solve the problem, we encode the task rules in Promela:

```
int N;
active proctype main() {
 N = 7;
 do
        ::(N % 2 == 0) -> {
         printf("n␣=␣%d,␣div␣\n", N);
         N = N / 2;
        }
        ::else -> {
         printf("n␣=␣%d,␣3n+1␣\n", N);
         N = 3 * N + 1;
        }
 od
}
ltl check_me { [] (N != 1)}
```

Model checking games

- We added the LTL rule check_me, it which we try to ensure that N will never be 1
- As the solution exists, the model checker while verification will find a path to get 1 from 7 using the rules
- The steps how to get 1 will be printed by our printf operators in the simulation mode using a generated counter-example trail.

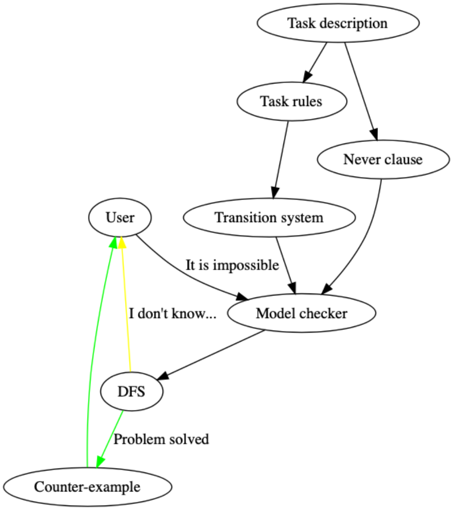
Model checking games

Formally, the model checking games can be represented as

$$\{\neg\phi, T\} \xRightarrow{\text{model checking}} ((C \subset T) \vdash \phi) \vee \emptyset \tag{2}$$

Where ϕ is an LTL formula, T is a transition system, C is a counter-example as a solution of the task, part of the transition system, $\emptyset$ here means that the verifier was unable to find a counter-example.

Model checking games



On implementation of fuzzy genome comparing during a model checking game

Fuzzy genome comparing during a model checking game

In order to do fuzzy substring search, we added the following into the Z-function Promela code:

- a non-deterministic choice when we compare symbols while building the Z-function;
- a condition to limit the possible percentage of changes.

Fuzzy genome comparing during a model checking game

So, the comparison process (instead of testing s1==s1) becomes:

```
bool isEquals = false;
if
 ::(s1 == s2) -> isEquals = true;
 ::(s1 != s2) -> isEquals = false;
 ::(s1 != s2) -> {
  casesTotal++;
  if
   ::(casesOk * 100 / casesTotal <= prob) ->
    { isEquals = true; casesOk++; }
   ::else -> isEquals = false;
  fi
 }
fi
isOk = isEquals
```

Fuzzy genome comparing during a model checking game

- Here *prob* is a given percentage probability with which we want to compare the strings
- This algorithm means that when constructing the Z-function, the equality of characters is tested and a deviation is allowed with a given probability

Fuzzy genome comparing during a model checking game

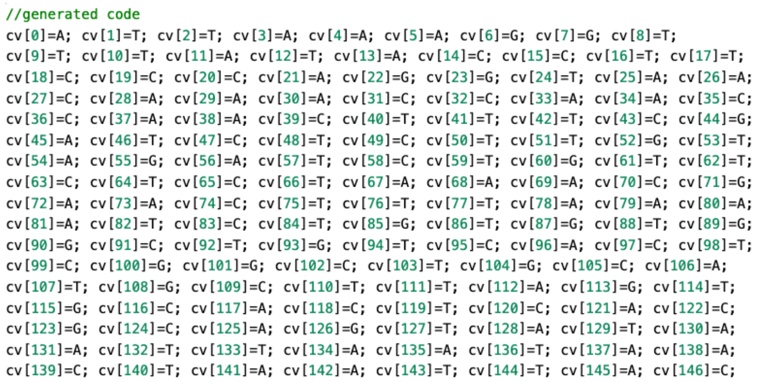
As input strings are represented as arrays, we add the definition for the input alphabet:

```
\#define A 0
\#define T 1
\#define G 2
\#define C 3
```

As Promela does not support I/O operations, we implemented a .fasta file (with input genome sequence) processor and a Promela code generator

Fuzzy genome comparing during a model checking game

Generated input code:



```
//generated code
cv[0]=A; cv[1]=T; cv[2]=T; cv[3]=A; cv[4]=A; cv[5]=A; cv[6]=G; cv[7]=G; cv[8]=T;
cv[9]=T; cv[10]=T; cv[11]=A; cv[12]=T; cv[13]=A; cv[14]=C; cv[15]=C; cv[16]=T; cv[17]=T;
cv[18]=C; cv[19]=C; cv[20]=C; cv[21]=A; cv[22]=G; cv[23]=G; cv[24]=T; cv[25]=A; cv[26]=A;
cv[27]=C; cv[28]=A; cv[29]=A; cv[30]=A; cv[31]=C; cv[32]=C; cv[33]=A; cv[34]=A; cv[35]=C;
cv[36]=C; cv[37]=A; cv[38]=A; cv[39]=C; cv[40]=T; cv[41]=T; cv[42]=T; cv[43]=C; cv[44]=G;
cv[45]=A; cv[46]=T; cv[47]=C; cv[48]=T; cv[49]=C; cv[50]=T; cv[51]=T; cv[52]=G; cv[53]=T;
cv[54]=A; cv[55]=G; cv[56]=A; cv[57]=T; cv[58]=C; cv[59]=T; cv[60]=G; cv[61]=T; cv[62]=T;
cv[63]=C; cv[64]=T; cv[65]=C; cv[66]=T; cv[67]=A; cv[68]=A; cv[69]=A; cv[70]=C; cv[71]=G;
cv[72]=A; cv[73]=A; cv[74]=C; cv[75]=T; cv[76]=T; cv[77]=T; cv[78]=A; cv[79]=A; cv[80]=A;
cv[81]=A; cv[82]=T; cv[83]=C; cv[84]=T; cv[85]=G; cv[86]=T; cv[87]=G; cv[88]=T; cv[89]=G;
cv[90]=G; cv[91]=C; cv[92]=T; cv[93]=G; cv[94]=T; cv[95]=C; cv[96]=A; cv[97]=C; cv[98]=T;
cv[99]=C; cv[100]=G; cv[101]=G; cv[102]=C; cv[103]=T; cv[104]=G; cv[105]=C; cv[106]=A;
cv[107]=T; cv[108]=G; cv[109]=C; cv[110]=T; cv[111]=T; cv[112]=A; cv[113]=G; cv[114]=T;
cv[115]=G; cv[116]=C; cv[117]=A; cv[118]=C; cv[119]=T; cv[120]=C; cv[121]=A; cv[122]=C;
cv[123]=G; cv[124]=C; cv[125]=A; cv[126]=G; cv[127]=T; cv[128]=A; cv[129]=T; cv[130]=A;
cv[131]=A; cv[132]=T; cv[133]=T; cv[134]=A; cv[135]=A; cv[136]=T; cv[137]=A; cv[138]=A;
cv[139]=C; cv[140]=T; cv[141]=A; cv[142]=A; cv[143]=T; cv[144]=T; cv[145]=A; cv[146]=C;
```

Fuzzy genome comparing during a model checking game

According to rules of model checking games, we should specify a negation for the rule that shows the fact of solving the puzzle. In a fuzzy genome sequence search, we specify a simple rule

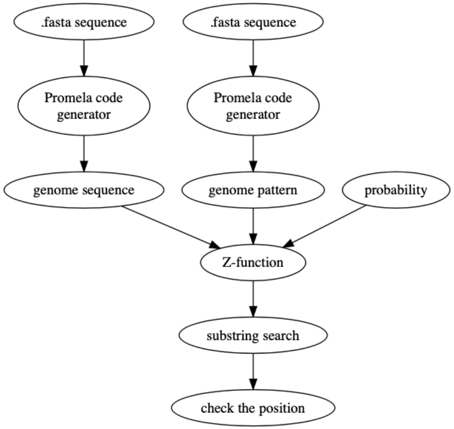
$$G(\mathit{impossible} == 1) \tag{3}$$

(”always it is impossible to find a substring”), where variable *impossible* is the variable that is changed in a linear substring search using Z-function we built previously

Overall structure of the solution

The model loads sequences, builds the Z-function and makes the substring search with the given probability of deviation. If found, the control variable is set. The model checking game here – to ask the model checker that it will never happen and its duty is to provide a counter-example with string substitutions after which the genome pattern can be found in the given genome sequence.

Overall structure of the solution



Results and their discussion

To start, we tried to do a search for a small pattern (less than 100 chars) with some mutations in a whole genome sequence string using a simple laptop with 8GB of RAM.

- After some runs in the SPIN simulation mode (it uses different random seeds) we were able to see that the substring is found
- Exhausted verification is not feasible due to huge memory and time consumption (as we have the non-deterministic choice and large state space)
- Bitstate mode (”-DBITSTATE”) with default parameters runs for some time and goes out of memory

Results and their discussion

- Hashcompact mode ("-DHC") with default parameters has a small memory consumption but finishes without producing any counter-example
- Swarm model checking with default swarm script runs out of memory (because it uses the bitstate mode).
- Swarm model checking using a patched swarm script to substitute the bistate mode to the hashcompact mode was able to produce a counter-example and solve the task

Results and their discussion

- To think further, to do a model checking game to compare two full genomes with a given deviation rate requires a lot of VTs with different randomized transitions seeds
- The task here should be divided into loading the data to common memory (the same phase to all VTs) and then different Z-function calculations using the same data. It would require a custom model checker
- We also see that the CPU swarm technique is not a good idea to execute a bunch of VTs, and possible GPU swarm or FPGA swarm should be used.