Model checking vesicular transport system

Ankit Shukla, Kuppaswamy Lakshman

Chennai mathematical institute

* ankit@cmi.ac.in * klachu@gmail.com

Abstract

Modeling biological is an art. Biological systems are very complex in comparison to the computer system that we currently can build, for example a human body is a perfect example of the complexity involved in the system. So in order to understand these complex systems, besides experimenting the best way is to make simpler elegant model with suitable abstraction of biological systems and try to encode them as a computer programs. To understand the biological system we need some additional machinary besides simulation and stastical analysis. This paper presents two such model and proposed method that can handle the scalibilty problems and prdict precise results as opposed to a statistical one. We use model chekers (Boolean satisficability, SAT) as a framework to encode and analyze these model. We claim that these tools are suitable for biological systems which require combinotorial analysis of the stated rules and with possibly large state space.

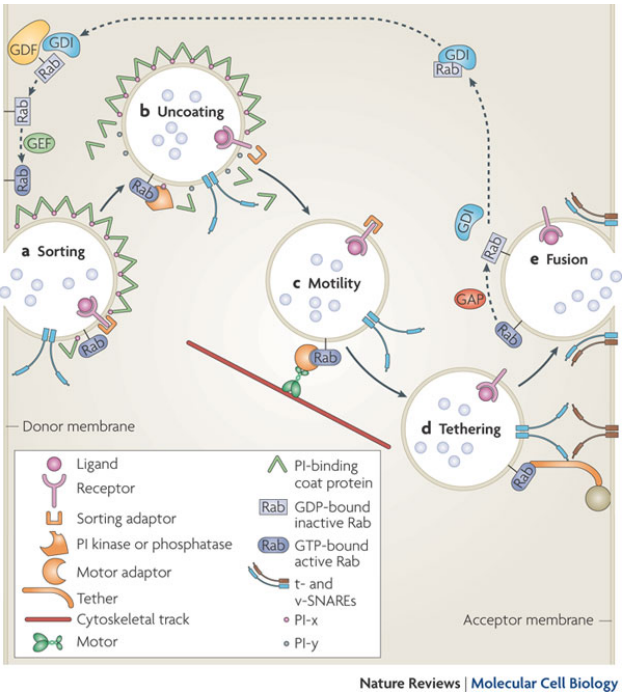
1. Fusion in eukaryotik cell

The process of vesicle traffic system is four steps linear process (Fig.1) involving:

1. Sorting of cargo molecules from the source compartment for packaging into the vesicle.
2. Budding of the vesicle.
3. Tethering of the vesicle to the target compartment.
4. Vesicle fusion.

We can understand the whole fusion process similar to that of a cargo transfer. First the sorting of the cargo to be shipped happens and then cargo left for the target on some transpoting structure and at the end shipped to the destination. Few points are trivially not clear such as how vesicle (cargo) finds (knows) its destination?

Fig 1: Mechanics of traffic. The sequence of events that take place during vesicle-transport and the role of the major classes of traffic molecules is shown here.



Nature Reviews | Molecular Cell Biology

Complete details of these process can be found at [1-6] and we'll assume that it's evident what is happening.

2. Introduction

Molecular interactions regulate the movement of transport vesicles between the membrane-bound compartments of eukaryotic cells [Rothman, 2002; Munro, 2004]. Rothman, Schekman and Sudhof were awarded the Nobel Prize for identifying key molecules involved in this process. These include: Arf and Rab GTPases whose presence or absence encodes compartment identity [Stenmark, 2009]; coat proteins that regulate molecular loading from source

compartments onto vesicles [Robinson, 2004]; and SNARE proteins that regulate fusion of vesicles into target compartments [Jahn & Scheller, 2006; Wickner & Schekman, 2008].

These molecules form the basis of an abstract representation of vesicle traffic [Mani & Thattai, 2016a; Mani & Thattai, 2016b]: compartments are the nodes, and vesicles are the directed edges, of a transport graph; different molecules move in cyclical fluxes along this graph; the properties of compartments and vesicles are defined by the molecules they contain.

In particular, we focus on the transmembrane SNARE proteins, which form a key part of the cellular addressing system. The fusion of a vesicle to a target compartment is driven by the zipping together of SNAREs [Jahn & Scheller, 2006; Wickner & Schekman, 2008]. In their fusion-competent state, SNAREs form a four-helix bundle: three components of this bundle are Q-SNAREs, the fourth is an R-SNARE. Both vesicles and their target compartments will contain SNAREs. Typically the Q-SNARE triple-helix is confined to one membrane surface, while the R-SNARE is confined to its fusion partner. (In the older literature SNAREs were often separated into vesicle v-SNAREs and target compartment t-SNAREs; however, it is now clear that SNAREs can act bi-directionlly, or even during homotypic fusion, so we use the more explicit Q/R classification.)

There are many types of Q-SNAREs and R-SNAREs spread between different vesicles and compartments. Only certain Q-SNAREs can zip with certain R-SNAREs, regulating precisely which vesicles fuse with which targets. Molecules such as GTPases and Sec/Munc proteins on vesicles and compartments can activate or inhibit SNAREs [Rossi et al., 2004; Stenmark, 2009]. For the purposes of our analysis we consider a Q-SNARE triple helix as a single complex, and the R-SNARE as its cognate partner. Crucially, the zipping of a single active SNARE complex on a vesicle with a single active SNARE complex on a compartment is necessary and sufficient for fusion [van den Bogaart et al., 2010]; we refer to this as SNARE pairing. Once a SNARE pairing drives a fusion event, the NSF protein uses the energy of ATP hydrolysis to split apart the four-helix bundle into its constituent Q and R components. NSF is present on all compartments of a eukaryotic traffic system, but is not found in appreciable quantities on vesicles.

3. Previous work of use of formal methods in Biology

Formal verification techniques are already used for aiding various aspects of biological research [11,13,16,15,22]. In particular, the robustness of models of biochemical systems with respect to temporal properties has been studied [18,6,19]. Our work is, to the best of our knowledge, the initial application of formal verification to studying the vesicle traffic regulatory networks and, as such, it opens up a novel application area for the formal verification community. As previously discussed, with respect to related studies in simulation based vesile traffic system [mani ad Thattai] our method can offer a higher degree of assurance, more accuracy, and better scalability than the traditional, simulation based approaches.

Computing has contributed to the biological sciences by making it possible to store and analyze large amounts of experimental data. Recently, the development and application of computational methods and models that capture key biological processes and mechanisms is increasingly utilized towards helping biologists in gaining a clearer understanding of living systems and improving predictive capabilities. The ability to effectively explore and analyze such biological models is key to making scientific progress in the field.

Cell evolution and color problem

4. Color Encoding problem

4.1. Problem statement

Given a valid vesicle traffic system in which all molecules are recycled (i.e. all paths are cyclic) and all vesicles are distinct (3-connected), what is the minimum number of coat proteins that are required. Abstractly to find the minimum number of masks (we'll connect the coat proteins to mask in the traffic system in future sections) that is required in order to satisfy vesicle traffic rules in steady-state condition (every leaving molecule is recycled).

4.2. Introduction

Eukaryotic cells are filled with membrane-bound compartments, each of which has a characteristic size, shape and protein composition. During the life of the cell, proteins are constantly being transported inside vesicular carriers from one compartment to another, yet the structure and make-up of each compartment remain more or less constant. This implies that there must be a balance between outgoing and incoming membrane traffic and, furthermore,

that the cell must be able to transport proteins selectively, discriminating between proteins that need to move on to another compartment and proteins that need to remain behind. Thus, the process of vesicle formation must be tightly regulated, both quantitatively and qualitatively.

Studies on clathrin-coated vesicles provided the first indication that the coat may serve two purposes: to deform the membrane into a budding vesicle and to select the contents of that vesicle. Our focus will be on the second.

We think the whole cell traffic network as a graph. In the topology of the graph, nodes are compartments of the molecules and molecules from one compartment to another travel via a smaller compartment (container) called vesicle. Vesicles carry molecules present on the source node and transfer them to the target compartment i.e only subset molecules present on the nodes are allowed to flow via vesicle. We can think this mechanism as molecule flow from compartments have some form of mask. We represent these vesicles as edges on the graph, where the direction of edge provide information about the source and target compartment and with every edge there is an associated mask. The whole vesicular system is a recyclable network. i.e every outgoing molecule is coming back to its original source node via a cycle. The problem is to come up with the minimized masks that is compatible with the valid system.

4.3. Overview of the model

There are two kinds of nodes in the system: compartments and vesicles. Compartments are all in steady state (recyclable). There are budding edges going from compartments to vesicles and fusion edges going from vesicles to compartments.

We describe coat proteins using two sets:

1. The set of molecules that the coat loads onto the vesicle (inclusion set, I)
2. The set of molecules that the coat avoids loading on the vesicle (exclusion set, E)

Whatever molecules from its inclusion set a coat finds on the source compartment, then it definitely loads it on the vesicle. If the source compartment contains molecules from the exclusion set of the coat protein, then these molecules are not loaded onto the vesicle. Since all molecules in the system are recycled, we identify molecules with the cycles on which they travel. In the model, we assign a single molecule to each elementary cycle of the traffic

graph. Vesicle and compartment compositions are then given by the set of elementary cycles of which they are a part. For each vesicle, we find the inclusion set (set of molecules that are present on the vesicle) and exclusion set (set of molecules present on the source compartment but not on the vesicle) for the coat responsible. We then define a coat incompatibility graph, where the nodes represent coats. Two nodes are adjacent if (I1 intersection E2) OR (I2 intersection E1) is not empty. The minimum number of colors in this graph represents the minimum number of coat proteins required in the system.

Mathematical view This as a black box procedure. By solving the matrix multiplication procedure A mult B where 2 - D matrix A represents the graph (1 in matrix : means inflow -1 means outflow , others 0) and other 2-D matrix represent whether a type of molecule present at which compartment. Equating to multiplication to get stability condition cause whole cause representing them as equations form resulting into a form that implies that for stability there has to be cycles, vesicle flow over a cycle is conserved.

4.4. Modeling

We modeled the system as a SAT problem by using bounded model checker CBMC. In comparison to simulation methods model checking turn out to very efficient way of solving this kind of problem. Rather than coding the properties directly we did some preprocessing to minimize the unwinding that CBMC (bounded model checker we have used) takes.

The input will be a graph where a node represents a compartment. Vesicle are moving out as edges. There is an inflow and outflow of the vesicles. With each possible edge, we assign a corresponding mask, that might not be unique.

Prob Statement: Find the minimum number of mask that is required to make sure that following conditions are met:

1. Graph is strongly connected.
2. Graph is Three Connected.

Steady state condition specifies that every leaving molecule come back to its source node via a cycle. This is the basis for recycling of SNARES.

We can state this specification regarding graph-cuts, that is each edge-graph-cut should maintain the inflow-outflow flux or as a property of assignment of molecules, such that each for each leaving molecule there is a cycle

where that molecule is present on every edge and hops taken. Required Specification is listed in Listing 1. For efficiency reasons we'll divide the whole case into two as stated in last chapter.

```
# Every edge E is represented by a label; labelE (a bitvector).
# E(x,y): Edge between node x and y. E.source = x, E.target = y
# labelE(m): mth molecule is present on edge E.
# Edge(m)(x,y): Edge between node x and y with labelE(m).
# seq(a1..an): each ak represents a node. len(seq) is |seq|.
# N: total number of nodes, = : Mathematical equality.

∀ z ∈ {Nodes} ∀ E ∈ {Edges}: E.source = z ⊃
    ∀ m ∈ {Molecules}: labelE(m) ⊃
       (∃ a seq(a1..an): (2 ≤ |seq| ≤ N)  ∧
            ((a1 = z) ∧ (a2 = E.target) ∧
             ((∀ k from 1 to |seq| − 1) ⊃  Edge(m)(ak,ak+1)) ∧
             (Edge(m)(an,a1)))
```

Listing 1: **Steady state specification.**

4.4.1. Mask

Now consider associating with each possible edge a mask. That will determine the molecule outflow possibility.

Lets make a graph that have 3 Nodes representing three compartments . And 1 - 3 (inflow to Node three from 1) 3 - 1 outflow from three to 1 3 - 2 outflow from 3 to 2 , 1 - 2 , 2 - 1 hence total 5 Edges and three Nodes. Three Cycles are preset. 1 - 2 - 1, 1 - 3 - 1, 1 - 3 - 2 - 1.

ADD DIAGRAN OF THE GRAPH

Rules : A Physical system system is allowed if : 1. Strongly connected. 2. Three connected.

There are substances that are present in at a compartment and few / all might be flowing through the a edges and hence outgoing substance has to be present at the compartment. So we require in this case at least log 5 $\simeq$ 3 substances. Represent the matrix for the substance as a binary switch whether its present in a loop or not. 1 1 1 would mean that that substance is present at all the three loops. Reduce the binary switch to a graph coloring problem and solve the same to find the minimum required color.

4.5. Analysis

Learning from our previous project where unwindings was really a problem and program was taking too much time before reaching into SAT solving

phase we represented graph as 1 dimensional array which have edge between n and n+1th element.

```
# G[2N]: Graph with width twice as number of nodes.
# Graph is 3 connected.
#define N 5
#define X 2 × N
...
unsigned int G[X];
```

Listing 2: **Graph encoding.**

This representation will cause additional book-keeping tasks but was very useful to decrease the loop unwinding in our code. In our Listing to illustrate the code with better readability we will assume that the graph is represented as a two dimensional matrix where G[i][j], G[i][j] = 1; there is an edge between node i and node j.

We represent nodes (compartments in the cell) in the graph as a bitvector of fixed length M (Total types of the molecules), representing information about the presence or absence of the molecule type at the ith place.

We fix the graph as input. Then have created a preprocessing by making the edge as a C programming language structure, that consists of each node having additional information attached to it which we can later use just by one lookup. This includes the source and target node i,j and the bitvector representing the edge (edgeWeight), the chosen mask and inclusion and exclusion set.

```
# Define a structure to represent the EdgeBag that is a tuple :
# edgeBag := (i, j, edgeWeight, mask, inclusionSet, exclusionSet
struct EdgeBag
 {
   int ith;
   int jth;
   bitvector edgeWeight;
   bitvector mask;
   bitvector inclusionSet;
   bitvector exclusionSet;
 };

....
# Create a structure that will contain edges.
struct EdgeBag edgeBag[len];
```

Listing 3: **Necessary condition property.**

But we still need and initiation of the structure. To initialize we just fix source and target node and leave everything else as it is; i.e all possible assignments.

```
# Set ith jth position and edgeweight to the position of
  edgeBag[len] array

edgePos = 0;
for(i=0; i<N; i++) {
    for(j=0; j<N; j++) {
          if (G[i][j] == 1) {
             edgeBag[edgePos].ith = i;
             edgeBag[edgePos].jth = j;
             edgePos = edgePos + 1;
          }
    }
}
```

Listing 4: **Necessary condition property.**

The plan is simple; we constraint these variable values such that the system rules are followed. We do this by use of a Boolean variables and make that variable as a part of query to make sure that the assignment given after a run will be compatible with the rules specified. This way we can check properties about the system by making sure that basic rules are followed.

4.5.1. Steady state condition

The steady state is similar to the vesicular chapter hence we'll skip the details. But recall the steady state condition or homeostasis requirement for the whole system means that each molecule leaving the node on a vesicle should come back to its source node in a cycle, i.e., for every molecule leaving the node there exists a cycle with that molecule present on each of the edges and nodes of the path taken.

4.5.2. Reduce the connection between the edges as 4 color problem based on inclusion and exclusion set

Next we specify the inclusion and exclusion set for each edge that is present in the graph. W make sure that the edge

```
# Specify inclusion and exclusion set for edge that is present
    in the graph.
# Add inside the condition of the mask structure possibility.

```

```
for(i=0; i < len; i++) {
      edgeBag[i].inclusionSet = edgeBag[i].edgeWeight;
      edgeBag[i].exclusionSet = nodes[edgeBag[i].ith] & (~ (
    edgeBag[i].ith)) ;

# Specify what are the possible configuration of the mask
    possible in the current setting.
# For any edge (mask & compartment config) == edge configuration
    .
     __CPROVER_assume (( nodes[edgeBag[i].ith] & edgeBag[i].mask
    ) == edgeBag[i].edgeWeight);
}
```

Listing 5: **Setting inclusion and exclusion masks.**

```
# Making edge between the present edges to make it a coloring
    problem
for(k=0; k<len; k++) {
    for(l=0; (l<len); l++) {
          if(k == l)
             colorGraph[k][l] = 0 ;
          else if ((( edgeBag[k].inclusionSet & edgeBag[l].
    exclusionSet) != 0) ||
                   ((edgeBag[k].exclusionSet & edgeBag[l].
    inclusionSet) != 0) ) {
                   colorGraph[k][l] = 1;
       }
      else
        colorGraph[k][l] = 0;
    }
}
```

Listing 6: **Reducing the mask problem to n-color problem.**

Solve four color problem to get the minimum color that is required to solve the prob.

Have significance to Biology as cell contained max three or four color.

Now we have to just solve the n color problem. There are many ways we can achieve simplest is shown in the Listing

```
# Apply the constraints for the finding minimum coloring problem
    .
# Constraint1: all nodes are of some color.
# assignColor(C): C determined how many colors are given and
  constraining C to be n we get whether its possible to restrict
     the
```

```
5    vesicle traffic system to be colored by n colors.
6
7  for(i=0; i<len; i++){
8        colorSet[i] = assignColor(C);
9  }
10
11 # Constraint2: If two edges are connected then both can't have
       same color.
12 C2 = 1;
13 for(i=0; i<len; i++){
14   for(j=0;j<len;j++){
15         if(colorGraph[i][j] == 1) {
16               C2 = C2 && (colorSet[i] != colorSet[j]);
17            }
18     }
19 }
```

Listing 7: **Necessary condition property.**

4.6. Comparison with the simulation method

Simulation method was able to handle the total number of nodes till N = 6. Scaling besides this limit was really a challenge. We checked every graph and achieved the same result and in some cases

5. Scalability and Discussion

We have made our model abstract enough to work with model checkers and used the graph's structural properties to reduce the search space. We follow an approach that makes our code more scalable by restructuring the code that which suits some common knowledge about the code scalability.

(i) Decrease formula size and number of clauses: Generally, it is the thumb rule that if you have reduced formula size you have a better chance to find a counterexample. But this is not true all the time. We have used the technique to reduce the formula size by tweaking our code and in some places constraining system to search in a restricted search space.

(ii) Avoid Variable coupling: Trying to minimize the number of connections or amount of interaction between variables helps a lot in a case where there are too many variables to solve for the SAT. Avoiding coupling is an important feature of the class of complex system we are trying to build, and it helped us in scaling this complex system up to 10 nodes.

6. Conclusion

We find that the number of coats required in the system is much lesser than the number of vesicles made in that system. The original goal was with the help of this model explain why only three or four coats colors are sufficient for the complex vesicle traffic system inside the eukaryotes.

We were able to be solve biological problems far efficiently and with higher assurance (it can enumerate the all possible rules) and scalability specially in the color encoding problem where we able to do far better in comparison to the simulation methods. In both cases we were not able to predict and explain the complete behavior due to reasons specified earlier. One possible reason may be we still do not know enough about these system in terms of details of reasoning and also may have skipped/missed few important details while modeling the system itself.

Most enthusiastic result of these project was some times model checkers performs far better in comparison to other times on some biological process and hence studying the important aspect that **why in certain case model checkers work might be connected to optimizations of the biology itself** is new path that future works can focus.

6.1. Introduction

Central problem of cell biology is to understand the working of cell growth (and shrink).

7. Cell evolution basics

SNARES are part of synaptic vesicle [Trimble, W. S. et al., 1988; Baumert, M. et al., 1989; Bennett et al., 1992] and their role in fusion mechanism is well established. They are proteins necessary for the fusion. We will focus on the mechanism of SNARE mediated fusion; a SNARE protein present on the vesicle membrane (v-SNARE) forms a complex with related SNAREs present on the target compartment (t-SNARE), resulting in docking the vesicle to the target compartment. One SNARE pairing in necessary and sufficient condition for the fusion [[Weber, T. et al., 1998]. SNARE traffic mechanism involves the following steps (Fig 1):

- Pack V-SNAREs into outgoing vesicles at the source compartment.

- Transport of these vesicles to their respective target compartments.
- Fusion of vesicles with the target compartments by SNARE complex formation .

SNARE complexes are resolved into individual SNAREs by the action of the cytosolic proteins NSF and alpha-SNAP; these SNAREs can then be used for another round of fusion [Jahn, R., & Scheller, R. H., 2006]. We know that the cell makes use of recycled SNAREs because blocking the reaction of NSF and alpha-SNAP eventually blocks secretion [Malhotra, V. et al., 1988]. But before the v-SNARE can be used in another round of fusion, it must first be sent back to its source compartment. Therefore, the picture we have so far of SNARE traffic is incomplete in the sense that it does not include the mechanism of recycle of v-SNAREs. In this chapter, we present a possible mechanism of SNARE recycle that makes use of the molecular machinery discussed so far, and the repercussions of this mode of SNARE recycle on the global topology of the vesicle traffic network.

Fig 2: Figure 1: Schematic representation of the mechanism of action of SNAREs and SNARE-interactors. (A) Sequence of events that lead to the fusion of a vesicle with the correct target compartment: The source compartment produces a vesicle which contains v-SNARE molecules. This vesicle travels to the target compartment where the v-SNARE encounters its corresponding t-SNARE. The v-SNARE contributes a single SNARE motif and the t-SNARE contributes the rest of the three motifs required for the SNARE complex. The SM protein specific to these SNAREs oversees the assembly of the v- and t- SNAREs into a SNARE-complex. SNARE complex formation releases enough energy to enable vesicle fusion. (B) Post-fusion, the v- and t-SNAREs remain bound as a stable SNARE complex. Resolution of this complex into individual SNARE molecules requires the action of the ATPase NSF and alpha-SNAP.

8. Cell evolution as Boolean Circuit

We are interested in modeling it as simple Boolean circuit [Mani-thattai] and understanding useful properties of the underlying system. Initial attempt has been made to model this Boolean circuit by using a simulation and taking average over the results. That experiment has been covered in Somya's [Mani] thesis completely. We have modeled the same system using the bounded model checker (BMC). We have used CBMC (C-bounded model checker) and abstracted away many details to make model as simple as possible and compute exhaustive solution.

8.1. Modeling as Boolean Circuit

Cell can be thought as a container of compartments (eucaryotic cells) where each compartment contains molecules. We fix different types of molecule present in the system, upfront. So total possible different compartment are 2^M. We are interested in how many different compartments are present and will not care about how many of similar compartments are there. So, a state will represent how many different compartments present in the cell. Doesn't care if two/more copies of the same compartment present. The system will start in an arbitrary state and using transition rules it will make a transition.

Considering a compartment as a subset of different types of molecules. The system is driven by two set of rules:

- Update Rules: Driven by an arbitrary M * M buddying matrix (absorption table).
- Delete Rules: Driven by an arbitrary M * M fusion matrix (release table).

In terms of biology the budding matrix specifies which form of vesicles are allowed to go out from a compartment. One obvious restriction on vesicle that is allowed to go out is that it has to be a subset of the molecules present on the compartment.

These tables will act as a rule for the update and deletion of the contents (molecules) from the compartments. There are total 2^{M*M} possible rules for given M * M matrix (table). In theory there is no restrictions on these tables beside a few obvious ones (eg. subset leaving requirement), but in practice we can assume that the budding matrix is constrained and at most two type of the vesicle is allowed to go out from a compartment. We impose no such restriction on the fusion matrix i.e for any vesicle coming to fuse the target compartment it is possible to so, if the matrix allows it. In both of these table, rows are compartments.

State update (one iteration): Lets assume we have a current state specifying the total number of different compartment that the cell have. In the next step, set of compartments release vesicles (set of molecules) as dictated by release table. Every compartment can absorbs all those vesicles that it is permitted by the absorption table. Vesicles absorbed by one compartment must have been released in this iteration by some other compartment.

These stages are concurrent: for each compartment C currently present, its behavior is determined by row C in both tables.

Creating new vesicles: To create a new vesicle there is more than one possibility. Each released vesicle that is not used, becomes a new compartment. All vesicles that are released but not used merge to form a single new compartment.

Need not stick to the absorption/release table format for state update. Can use any update rule that takes set of subsets of molecules to a new set of subsets.

Optimizations that make SAT solution go through may have interesting implications for biology, so useful to experiment with such optimizations regardless of their current biological significance.

New interesting queries: Count the number of equilibrium states for a given set of rules.

8.2. Synchronous and asynchronous Model

8.2.1. Synchronous model

Cell starts in the arbitrary initial state and the evolution is dictated by the two one step rules : update and delete. These update/delete can be performed in the on by one fashion who dictates the next state to achieve a specific state which of our interest.

8.2.2. Asynchronous model

Rather than performing the task one step using either update or delete in one by one fashion we can allow the cell to be updated in asynchronous way and hence the next state is dictated by the way these two steps are performed.

Main Goal of David Dills Asynchronous Model was : To define timing robustness as the ability of cells to function correctly when there are significant variations in the timing of internal events, and explore timing robustness in cellular control systems using symbolic model checking.

8.3. Implementation

8.3.1. Basic setup

We model cell evolution as a transition function. Bounded model checker is very suitable for these type of encoding where we can check the property by fixing some lower bound and incrementing it thereafter to achieve the desired goal.

Lets fix different types of molecules present in the system (N). Hence the total number of possible different compartments can be 2^N lets call it M (power-set of N). As we consider only distinct compartments we can represent the state of the system as M bit length bitvector where each of the bits represent the different ith molecule and 0 or 1 represent their present or the absence.

8.3.2. Example

Lets Fix N = 2. Total number of molecules are two. Let's represent them as two different natural number 1,2. Total number of possible combination of N = $2^2 = 4$ will be represented as

- {0} : Represents Nothing is present. Compartment is empty.
- {1} : Represents A compartment with only one molecule 1 is present.
- {2} : Represents A compartment with only one molecule 2 is present.
- {1,2} : Represents a compartment with both the molecules are present.

8.3.3. Transition

Starting from an arbitrary initial state and evolving (making progress from initial state) by using a transition function based on two tables

1. Buddying Matrix : removes the materials from the cell compartments or completely delete a it (equivalent to a empty compartment).
2. Fusion matrix : adds some material (molecules) to the current compartment of the cell or adding new compartment.
    - Prev State: {{1}, {2}} Next State: {{1,2}, {2}}. Added molecule {2} to first compartment.
    - Prev State : {0}: dead. Next state: {1,2} : Cell with a compartment having two molecules, 1 and 2.

8.3.4. Table representation

Both table have M * M dimension. M is total number of subsets for a given N. For N = 2 i.e total $2^2 = 4$ subsets, hence M will be 4. We constraint the table to take only Boolean values and leave the value arbitrary, hence the possible values at any place of 2-dimensional table is either have 0 or 1. Value 0 represent no possibility to make a move. 1 represent move is allowed.

Rules for Update and Deletion Tables del[M][M] , update[M][M]:

Compartment	00	01	10	11
00	0	1	0	0
01	0	0	0	1
10	0	0	1	0
11	0	0	0	1

Table 1: Update table

1. For Update Table :
   - Exactly One possibility in a single row to make progress.
   - Basic Example : Update caused a compartment {1} to add molecule 2. Hence become compartment {1,2} in next state
   - Progress might be just staying there; i.e no molecule is added to the current compartment.
   - Like $\{1\} \mapsto \{1\}$ after an update state means compartment remained the same.
2. For Deletion Table :
   - At least one possibility in a single row for the deletion table.
   - Allow multiple possible deletes.
   - This might be used to model the situation based on non-deterministic choice.

Lets see with the help of an example what are the possible cases. Lets analyse the results of a single update table rule.

1. update[1][3] = 1 : means second row and forth column is = 1 and hence a valid move. As the rules are deterministic and there is only one choice if a compartment has this configuration {1} in next state molecule 2 will be added to this compartment and hence it'll become {1,2}.
2. update[2][2] = 1 : means third row and third column has 1. Means in an update phase a molecule with configuration {2} will remains {2}. Nothing is going to be added. Stay on.

Adding same molecule to the current compartment where already that molecule is present would have no change in configuration of the molecule.

Compartment	00	01	10	11
00	1	0	0	0
01	1	1	0	0
10	1	0	0	0
11	1	0	1	0

Table 2: Delete table

For example if we add molecule 1 to a compartment with configuration {1,2} its still be {1,2}. That means we are not taking care of count of same molecules present.

For the delete table lets see what possibilities are there.

1. delete[0][0] = 1 : if molecule is dead nothing can be deleted from it.
2. delete[3][0] = 1 , delete[3][2] = 1 : means if compartment has this configuration {1,2} in deletion phase there will be a non-deterministic choice between these two options.
   - delete[3][0] will give an effect (means) if compartment has configuration {1,2} before a delete phase in deletion phase this compartment will loose both elements and will become dead. i.e {0}
   - delete[3][2] = 1 : It'll give an effect (means) if compartment has configuration {1,2} before a delete phase in deletion phase this compartment will loose one element {1} and will become a compartment with only one molecule 2, {2}.

Another assumption: In order to make the behavior consistent initially we enforce that only last choice will be taken i.e getting an effect that only last delete is present. Deletion will be carried out.eg: from {1,2} two possibilities:

- delete {1} and hence new compartment will have {2} remaining.
- delete Both molecule and hence cell becomes dead to {0}.

In this case b will be taken as the choice.

8.3.5. Encoding used:

Implementation of the functionality of the table: We have encoded the table functionality as a go-to table. Rather than using table to state which things can be deleted from the table we have encoded "what happen if something is deleted from the table". Meaning if" let's row 3 (representing {1,2}) and column 2 (representing {2}) have a value " 1 " i.e delete[i][j] == 1; from a compartment with {1,2} deleted 1 and we now left with a compartment with a only one element/material {2}. Which is kind of abstract meaning rather than a explicit form. This saves few additional computation that is required after we know which can be deleted. Intuitively we might have encode this as delete[3][2] == 1 means from {1,2} and {2} you delete {2} and somehow figure out (via some calculation here trivial) to delete {2} from this compartment and transform this compartment to a single element compartment {1}.

States representation: States are represented using a Bitvector of length M. That means let's say N = 2 which have total possible $2^2 = 4$ those are {0}, {1}, {2}, {1,2}. state: 0100 (read from left side i.e from MSB) represent in meaning only one compartment present i.e with element/material 1 ({1}).

Similarly 0111 means three compartment currently present which are {1}, {2} and a {1,2} these are represented just by position (MSB rather than LSB yes it's unintuitive but will allow us to use table representation as a bit vector as not possible in any form to tell computer what {1,2} really mean). First zero represent compartment with no element present {0} not mean much. First 1 represent a compartment with single element 1 ({1}). etc last one i.e LSB 1 rightmost one represent presence of a compartment with two elements {1,2}.

The basic code is a loop which allows transitions to be made and then checks property of interest. Listing 1 shows the coded loop with the property that it will check for the lassos in the transitions system (using the variable state, next_state,looped,on_loop and with the help of transition() function able to make transition. __CPROVER_assert has been used to check the property of interest after one transition has taken place.

```
1 while(1){
2     save = nondet_bool();
3     if (save && (!saved)) {
4         l2s_s = state ;
5     }
```

```
6       on_loop = (save || saved);
7       next_state = transition(state, getRel, getAbs);
8       counter++;
9       state = next_state;
10      saved = on_loop;
11      looped = (saved && (state == 12s_s));
12      if (on_loop && !looped) savecnt = counter;
13      __CPROVER_assert( (!(on_loop) ||  (!looped || savecnt >
    10)), "every stable state is reached within 10 iterations");
14
15  }
```

Listing 8: **Basic loop handling the transition with checking the property after one update.**

State will make transition based on the transition function which will take help from delete and update to make a transition to next state. As it is possible to have a multiple one's in a row in deletion table we have to use some means to implement our most weighted delete a choice, here used that facility as getlastone() function. As the state representation is reverse i.e from MSB to LSB getlastone will mean implement most weighted delete. Which make the encoding consistent (we can think this as exactly one).

8.3.6. Updated Model

We employ few basic steps from our learning from the synchronous model. One it was too far from reality and we need to create abstraction to get away from all the required details and focus on large viewpoints. The basis of the project was to define an asynchronous model of the cell structure evolution by choosing right abstraction and verify properties to understand these system better, and if possible match against the experiment results in the labs.

Underline:

1. We can abstract away from the concept of vesicle which is the deviation from previous model where we were differentiating with the compartments and the vesicle.
2. Cell now have only compartments which grows and shrinks.
3. Compartment can either fuse or breakup.
4. Only pairs fuse and breakup partitions the compartments in two different molecules.
5. Rather than just avoiding the identical compartments we enforce consideration of conservation of molecule.

6. We'll take count of no.of similar molecules present. Count will be relevant in case when there is total only one molecule (count = 1 for a molecule).
7. An asynchronous model of evolution will take place, asynchronous in terms that choice for which one you pick for an update or delete.
8. As we can implement it non-deterministically the probability of the choice between fuse and breakup are equal. So we bias the probability by a fixed rate to favor of fusion. eg : prefactor * No of distinct pairs allowed to fuse.
9. In case where total compartments are less than 4 we'll always do breakup and in case where total compartments are ≥ 100 we always do a fuse. So restricting the focus set.
10. New compartments can be created via the breakup subset never get absorbed and leading to a creation of new compartment.
11. We have to find some evolution that is reproducible such that ordering of events does not matter. i.e the cell starting in certain configuration will end up in one configuration whichever way the ordering happened.

Two things still not clear in my mind.

In which sense he used the word molecules, it has a vesicle feeling? Multiple donor thing I think he meant more than a one donor creating a new compartment (forgot why that was so important).

8.4. Conclusion

We have experimented with the synchronous update model and found that it was not very scalable.