Stringmol Worksheet for the RUTSAC Workshop

August 5, 2011

1 Introduction

The Stringmol Artificial Chemistry (AChem), is a string based programming language. In Stringmol, 'molecules' are a string of characters. The Stringmol alphabet has 33 characters: 7 'functional' characters {`$>^?=}%`}, and 26 'non-functional' characters $\{A - Z\}$.

Stringmol molecules are initiated in a 'bucket', with a physics engine allowing molecules to collide and a flux of energy into the bucket allowing the molecules to react. There is also a decay rate for molecules in the bucket that sets a direct evolutionary pressure on replication.

Molecules in the bucket can be in one of two states, either they exist as a bound pair, or as a free floating molecule. Upon collision, molecules have an opportunity to bind; if binding test is successful, the molecules are aligned based on their best (complimentary) matching sub-sequences. When entering a bound state, the molecules will have a chance to execute their microprograms. Both molecules initiate their pointers (read, write, instruction, flow) on their respective strings, at the start of the alignment. The molecule with the most characters before the alignment is designated the 'active molecule'. The active molecule pointers are live at the beginning of the reaction and the instruction pointer executes one instruction per timestep. Instructions involve moving pointers to execute a program. The movement of pointers includes 'toggling' to pointers on the passive string; it is even possible to toggle all the pointers to the 'passive' so that it is executing its program.

In this worksheet, you will be introduced to the Stringmol web resource. The resource has two main sections: One runs a bucket of Stringmol molecules based on the initial molecules that you can specify. This allows you to observe the dynamics of any system you can design, with the mutation option turned on you can even watch systems evolve. The other section lets you react two strings, observing binding, alignment and step-by-step execution of the programs, allowing a full grasp of how two molecules react.

2 The replicase molecule

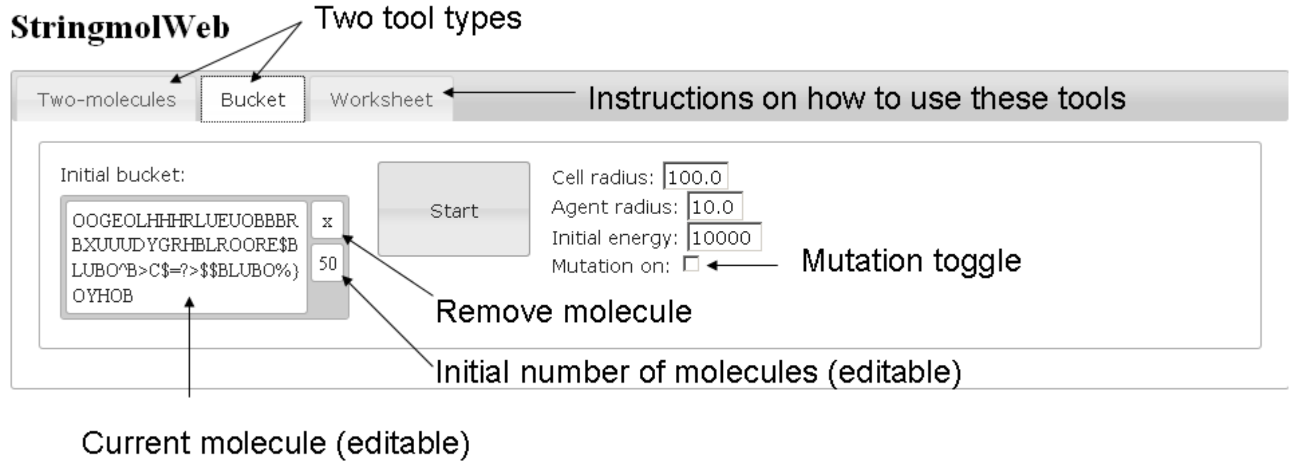
In this section we will introduce the a 'replicase' molecule. The molecule has been hand written, such that in a reaction with another copy of itself, it will produce an additional replicase molecule (2 replicase → 3 replicase). We will begin at the level of system dynamics, putting 50 replicase molecules into a bucket and observing how the system progresses. We will then go on to examine how the replicase molecule works as well as seeing how to break it.

2.1 The original replicase

The published results of Stringmol are all from evolving 'bucket' of replicase molecules. We aim to give you an in depth understanding of the system and an ability to observe a mutating system and have a general understanding of the types of reactions that are happening. In order to understand how a system is being changed, it is important to be familiar with normal function, which is why we begin with a description of the original replicase that can be used as the seed molecule.

We begin by observing the dynamics of multiple copies of the replicase molecule. Figure 1 shows the detailed layout of the 'bucket' tab, naming particular features and showing their locations. The named features will be used in the step-by-step instructions in this section.

Step by Step Instructions


StringmolWeb
Two tool types
Two-molecules
Bucket
Worksheet
Instructions on how to use these tools
Start
Mutation toggle
Remove molecule
Initial number of molecules (editable)
Current molecule (editable)

Initial bucket:

OOGEOLHHHRLUEUOBBBR
BXUUUDYGRHBLROORE$B
LUBO^B>C$=?>$$BLUBO%}
OYHOB

x

50

Cell radius: 100.0
Agent radius: 10.0
Initial energy: 10000
Mutation on: ☐

Figure 1: Bucket tab. Showing the locations of key features, such as the mutation toggle.

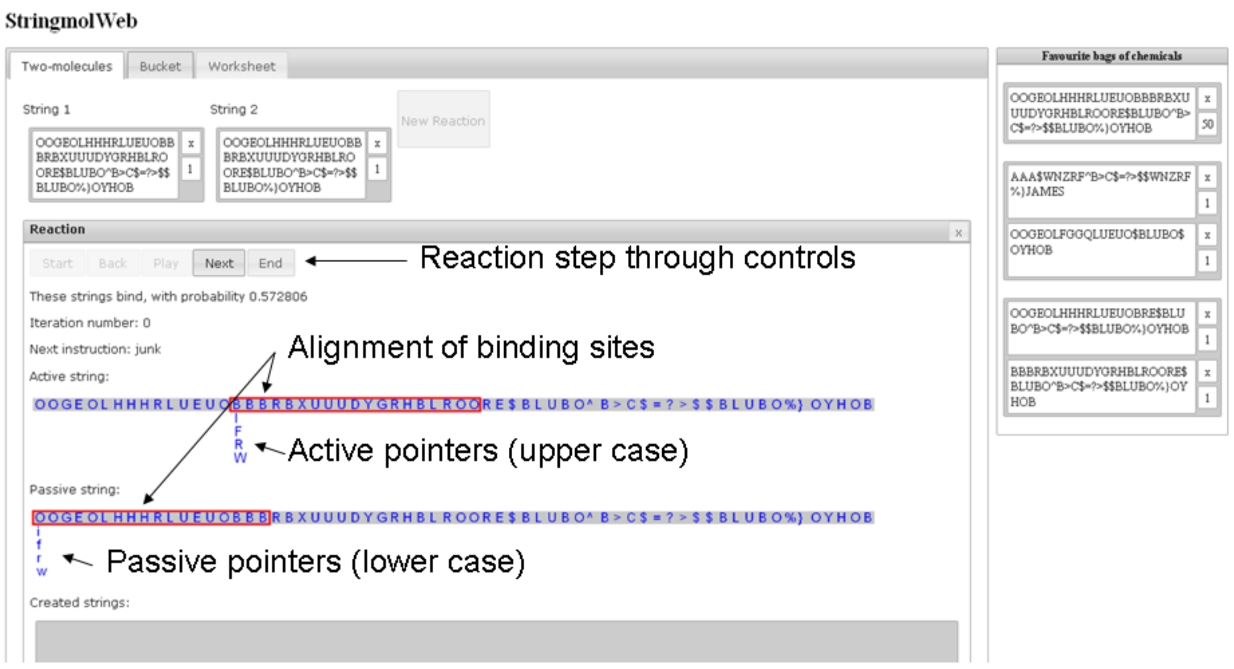
1. In the 'bucket' tab:
2. Turn off mutation by un-checking the 'mutation on' tickbox.
3. Put 50 'original replicase' molecules into 'the bucket'. You can drag the 'original replicase' string from the 'Favourite bags of chemicals' section and edit the initial number of molecules by clicking on the number and changing the value. You can also copy and paste the molecule, by editing the current molecule (by left clicking on it).

   Original replicase:

   `OOGEOLHHHRLUEUOBBBRBXUUUDYGRHBLROORE$BLUBO^B>C$=?>$$BLUBO%}OYHOB` Initial number: 50
4. Delete any other molecules that are in the 'Initial bucket' by clicking on the cross.
5. Check that your screen now looks similar to 1.
6. Click on the 'Start' button.
7. Observe the graph of the bucket's population starts at 50 molecules and rises to approximately 300 (where it will remain at a noisy equilibrium) The legend on the graph should only show one species: 'species 1'.
8. Press the 'Stop this bucket' button.

2.2 The two molecule tool

The two molecule tool, shown in figure 2 can be used in two ways, you can use it to step through part or all of a reaction to understand how the program executes. This can be done by following the pointer locations as the program is iterated. An example of this is given in section 4 for a self replicase string. The example explains the reaction in full: a valuable resource for those interested in writing their own programs in the Stringmol language. For the majority of this worksheet, a full understanding of the programming language and how programs are constructed in it is not required. The two molecule tool has an 'end' button, this skips to the end of the program (if the program has terminated) or as far along the program as has been run. Note: it is possible to construct infinite loops in Stringmol, so some programs will never produce an output (in a 'bucket', molecules trapped in an infinite loop will eventually decay and exit the system). If the program you are investigating does terminate you will be able to view the product in the 'Created strings' box, just below the reaction. Some reactions do not produce a string, it is possible (although we do not supply an example) to edit one or both of the strings that go into the reaction and produce new strings in that way. It is also possible that strings execute the programs, but the programs terminate having executed no meaningful instructions. Reactions with no products have significant effects on the 'bucket' system - each time step executing a reaction with no product is time that the two molecules involved are unable to bind with other molecules in the bucket and perhaps be replicated.


StringmolWeb
Two-molecules
Bucket
Worksheet
New Reaction
Reaction
Start
Back
Play
Next
End
Reaction step through controls
Alignment of binding sites
Active pointers (upper case)
Passive pointers (lower case)
Favourite bags of chemicals

String 1

OOGEOLHHHRLUEUOBBBRBXUUUDYGRHBLROORE$BLUBO^B>C$=?>$$BLUBO%}OYHOB 1

String 2

OOGEOLHHHRLUEUOBBBRBXUUUDYGRHBLROORE$BLUBO^B>C$=?>$$BLUBO%}OYHOB 1

These strings bind, with probability 0.572806

Iteration number: 0

Next instruction: junk

Active string:

O O G E O L H H H R L U E U O B B B R B X U U U D Y G R H B L R O O R E $ B L U B O ^ B > C $ = ? > $ $ B L U B O %} O Y H O B

I F R W

Passive string:

O O G E O L H H H R L U E U O B B B R B X U U U D Y G R H B L R O O R E $ B L U B O ^ B > C $ = ? > $ $ B L U B O %} O Y H O B

i f r w

Created strings:

Favourite bags of chemicals

OOGEOLHHHRLUEUOBBBRBXUUUDYGRHBLROORE$BLUBO^B>C$=?>$$BLUBO%}OYHOB 50

AAA$WNZRF^B>C$=?>$$WNZRF%}JAMES 1

OOGEOLFGGQLUEUO$BLUBO$OYHOB 1

OOGEOLHHHRLUEUOBRE$BLUBO^B>C$=?>$$BLUBO%}OYHOB 1

BBBRBXUUUDYGRHBLROORE$BLUBO^B>C$=?>$$BLUBO%}OYHOB 1

Figure 2: Two molecules tab. Showing the locations of key features

2.3 Breaking the replicase

Now that you have been introduced to the tools and the basics of how the original replicase works, we start to extend your understanding by demonstrating some single character mutations that can occur and how they can effect the replicase performance.

In biological systems, some mutations have little or no effect and others have a very large effect. Here we will look at fatal single character change to the replicase. We shall remove the ability to copy by replacing the '`>`' with an '`A`'. '`>`' is the instruction that moves the instruction pointer, implementing the copy loop - a vital operation for a replicase. As a consequence the copy will not happen and so the string will simply decay out of the system.

Step by Step Instructions

1. In the 'bucket' tab:
2. Put the broken replicase into 'the bucket'. You can do this by dragging the 'broken replicase' from the 'favourite bags of chemicals' section, or by editing the original replicase replacing the '`>`' with an '`A`'. (Or you can copy the string below and paste it over the current molecule.)

   `OOGEOLHHHRLUEUOBBBRBXUUUDYGRHBLROORE$BLUBO^B>C$=?A$$BLUBO%}OYHOB` Initial number: 50

   | Changed character
3. Click on the 'Start' button.
4. Observe the graph of the 'species 1' at 50 molecules and quickly falls to 0 (where it will remain). The legend on the graph will also show 'species 2' with a population level peaking at about 450 (±100). This chemical is a single character: the letter 'O'.
5. Press the 'Stop this bucket' button.

So, what is going on in this situation? Now that we have seen the dynamics we can apply the two molecule tool to discover how this string works and where 'species 2' comes from.

Step by Step Instructions

1. In the 'two molecule' tab:

2. set the broken replicase as ‘string 1’ and ‘string 2’

   `OOGEOLHHHRLUEUOBBBRBXUUUDYGRHBLROORE$BLUBO^B>C$=?A$$BLUBO%}OYHOB`

3. press ‘new reaction’.

4. press ‘end’ and observe that the final product of the reaction is an ‘O’ (the product of a reaction appears in the created strings box located underneath the active and passive strings). ‘End’ takes you as far along the reaction as has been calculated, so you will need to wait until the reaction has finished processing before you can see the product.

This shows why the broken replicase dies out of the bucket (and why a lot of ‘O’ chemicals are created in the process). When two broken replicases bind, the copy does not happen properly (because the copy loop is broken). Instead of looping and copying each symbol of the passive string, the program only copies the first character (which is an ‘O’).

3 A bucket full of replicases

The first task will be to set up and run a replicase simulation without mutation. The second task to check the viability of a new replicase (designed in the previous section) as a self replicase. Having familiarised yourself with how to set up, edit, run and stop simulations, we move onto ‘Invasion when rare’. Placing a small number of the new replicase molecules in the a bucket with a large number of the original replicase molecules, you will observe a ‘sweep’, as the new replicase establishes dominance (this is a stochastic take over, so it may not work every time) over the original replicase by an evolutionarily selectable advantage in its binding site. You will then check that the original replicase can not re-invade the bucket, by reversing the intital set up for the invasion when rare trial.

3.1 A Bucket of the original replicase

Use the bucket simulator (with mutation turned off) to watch the original replicase maintaining a population of (copies of) itself.

Step by Step Instructions

1. In the ‘Bucket’ tab:

2. Turn off mutation by un-checking the mutation tickbox.

3. Put 100 original replicase molecules into ‘the bucket’. You can drag the replicase string from the ‘Favourite bags of chemicals’ section and edit the initial number of molecules by clicking on the number and changing the value.

   Original replicase:

   `OOGEOLHHHRLUEUOBBBRBXUUUDYGRHBLROORE$BLUBO^B>C$=?>$$BLUBO%}OYHOB` Initial number: 100

4. Remove any other molecules (or mistakes) by clicking on the cross just above the number

5. Press the ‘Start’ button. This will begin the simulation, you should see the graph of the number of molecules changing.

6. Observe the graph and note that: species 1 reaches approximately 300 molecules.

7. Stop the simulation by pressing the ‘stop this bucket’ button.

The reason the population levels out at approximately 300 is to do with the flux of energy into the bucket, the decay rate and the time it takes to copy the replicase. If the energy flux were higher, the carrying capacity of the bucket would be greater. If the replicase molecule was made longer, the carrying capacity would be lower as each molecule would take more time to replicate. If decay was less likely the carrying capacity would be higher.

3.2 Testing the viability of a new replicase

We introduce a single character alteration to one of the replicase binding sites, as could easily happen in a mutating system. We will confirm that this new string is a valid self replicase. We will then go on to see what happens when these two species are in the same bucket - which one will survive and why?

Having used the bucket simulator with the original replicase, now try the 'invasion replicase' in the bucket on its own to make sure it is a viable self replicase.

Step by Step Instructions

1. In the 'Bucket' tab:
2. Remove the contents of the bucket. You can either set the molecule count to 0 by clicking on the number to the right of the string and editing it to be 0; or click on the cross in the above the number to remove the string more permanently.
3. Put 100 of the 'invasion replicase' molecules into 'the bucket'. You can drag the replicase string from the 'Favourite bags of chemicals' section. (You could instead copy the string from the text box of the 2-molecule simulator and paste it over the contents of a molecule already in the bucket.)
4. Remove any other molecules (or mistakes) by clicking on the cross just above the initial number.
5. Press the 'Start' button.
6. Observe that the 'invasion replicase' is viable and maintains approximately 300 molecules.
7. Stop the simulation by pressing the 'stop this bucket' button.

3.3 Invasion when rare

Put two of the 'invasion replicase' into a population of the original replicase and see how it takes over.

Original replicase:

```
OOGEOLHHHRLUEUOBBBRBXUUUDYGRHBLROORE$BLUBO^B>C$=?>$$BLUBO%}OYHOB
                            |
```
Initial number: 300

Marker of difference

Invasion replicase:

```
OOGEOLHHHRLUEUOBBBRBXUUUDYGRJBLROORE$BLUBO^B>C$=?>$$BLUBO%}OYHOB
```
Initial number: 2

Step by Step Instructions

1. In the 'Bucket' tab:
2. Set the number of invasion replicase molecules to 1 by editing the initial number in the bucket.
3. Add the original replicase to the bucket by dragging an dropping from the 'Favourite bags of chemicals'. Set its initial number to 250.
4. Remove any other molecules (or mistakes) by clicking on the cross just above the initial number.
5. Check that the invasion replicase will be plotted, by checking that the 'Ignore species with fewer than X molecules' is set to 1. (This can be done after the simulation is completed, by making the adjustment and re-plotting the graph.)
6. Press the 'Start' button.
7. Observe that the invasion replicase, despite its numerical disadvantage, replaces the original replicase (if it does not, try re-running the simulation several times as the takeover is stochastic).
8. Stop the simulation by pressing the 'stop this bucket' button.

This demonstrates 'invasion when rare', a phenomena observed in biology. Invasion when rare is based on the strong evolutionary advantage of one strain or species over another. Invasion when rare is common in a mutating system, there is a characteristic 'sweep' associated with the phenomenon. The dominant replicase is replaced by the invasion replicase in a characteristic time of about 55k +/- 10k timesteps. It is also possible that a new replicase with an evolutionarily selectable advantage will simply decay without being copied.

To check the necessity of the evolutionary advantage, you can put one of the original replicase into a population of the invasion replicase and observe that it never takes over. If you have just completed the instructions in this section, you can simply edit the initial numbers for the two species of replicase and start the simulation and stop it when the original replicase has died out.

3.4 Random walks

We have seen a 'sweep', where one replicase replaces another by an evolutionarily selectable advantage. We will now set up a situation where there is no evolutionary advantage. An easy way to set up this situation is to take the original replicase and make a variant by changing one character in the 'junk region' (characters that are not part of any functional or binding domain). This is another common single character mutation and effect that can be seen in an evolving bucket.

Step by Step Instructions

1. In the 'Bucket' tab:
2. Set the number of original replicase molecules to 150.
3. Add in a 'neutral mutation replicase' from the 'favourite bags of chemicals'. Or alternatively, create this neutral mutation replicase via the following steps: Add a second chemical to the bucket and edit it, pasting in the sequence for the original replicase. Alter one character in the junk region. The junk region ends at the first '$' character. Select the character immediately to the left of the '$' and change it to a different letter.

   ```
   OOGEOLHHHRLUEUOBBBRBXUUUDYGRHBLROORE$BLUBO^B>C$=?>$$BLUBO%}OYHOB initial number: 150
                                      |                             Marker of difference
   OOGEOLHHHRLUEUOBBBRBXUUUDYGRHBLROORA$BLUBO^B>C$=?>$$BLUBO%}OYHOB initial number: 150
   ```

4. Set the initial number of the variant you have just made to 150.
5. Press the 'Start' button.
6. Observe that the original replicase and the new variant population levels rise and fall and eventually one of them falls to a population level of 0. This process is stochastic, so if you don not see one of the population levels fall to 0 then do not worry. You should be able to observe the two chemicals performing a correlated random walk (with noise).
7. Stop the simulation by pressing the 'stop this bucket' button. You may wish to run this several times to check the 'winner' of the two replicase molecules is random. The duration of the random walk is highly variable.

When examining graphs of simulations where mutation is turned on, you will observe numerous 'sweeps' and should also be able to identify random walks. Some rare and more interesting phenomena (hypercycles) also have distinctive population dynamics, others are more subtle and cannot be readily identified from looking at the population graphs.

4 How does the original replicase molecule work?

We will use the 'two molecule' tool to understand at the program level how the molecular microprograms actually work. A full description of the language can be found in the technical specification (under the 'resources' tab on the website). Here, we will focus on key parts, gaining a general understanding, rather than in depth knowledge of the minutiae. We will use the two-molecule simulator to watch the original Stringmol replicase copying (another instance of) itself.

steps 1 to 20, stepping through the binding site and junk

step 21: the character `$` is the instruction character for 'search'. The instruction uses `BLUBO` as the query. The search instruction uses the contiguous block of non-functional characters immediately to the right of the `$`. The string `BLUBO` is an exact complementary match for `OYHOB`. The result of a successful search is to move the flow pointer (capital F below the string), to the end of `OYHOB`.

steps 22-26: the instruction pointer stepping through `BLUBO`

step 27: The character `^` is the toggle character. It can be modified by a single non-functional character to refer to each of the four possible pointers. In this case the instruction including the modifier is `^B` which means that it is the 'read pointer' that is toggled (toggling refers to switching which molecule has the active pointer, the four pointers can be toggled independently of each other). The active read pointer is shown by 'R', while the passive read pointer is shown by 'r'. After execution of this step note that the upper case R and lower case r have switched strings.

step 29: The character `>` is a pointer-moving instruction. Pointers are always moved to the active flow pointer 'F'; which pointer is moved depends on the modifier character. In this case `>C`, means that the write pointer 'W' is the pointer that is moved to the flow pointer.

step 31: The `$` with no non-instruction characters. This is effectively a null search, the result is that the flow pointer is moved to this position.

Step 32: The copy operator `=`, copies the character from the active read pointer 'R' and pastes it at the active write pointer 'W'. Both 'R' and 'W' are iterated one position.

Step 33: The `?` is an if statement. It is asking if the read pointer R is currently positioned within the string (as opposed to being off the end of the string). If it isn't the instruction pointer is skipped one character

Step 34 the `>` with no modification characters moves the instruction pointer to the position of the flow pointer +1.

Steps 35-221 continues around the copy loop.

Step 222: The escape condition is reached when the instruction pointer is on `?` and the read pointer 'R' is off the end of the other string. This moves the instruction pointer over the `>` allowing it to escape the loop.

Step 223: `$` without a modifier moves the flow pointer to the current location

Step 224: `$BLUBO` moves the flow pointer to the end of `OYHOB`. This is the end of the original string that entered the reaction.

Steps 225-229 steps through the non-functional characters `BLUBO`

Step 230 `%` is the cleave operator. This cuts the string in two where the flow pointer is currently located. Note that the product of the reaction appears in the 'Created Strings' box.

Step 231 `}` is the terminate character. The reaction ends when it is executed (returning the two unbound strings to the bucket).

5 Replication without reciprocation (Parasites)

The new replicase molecule, introduced in the previous section, was 'better' than (had an evolutionary advantage over) the original replicase. When the new replicase binds the original replicase, the new replicase is always the passive molecule, meaning that another copy of the new replicase will be produced as a result of the reaction. The reason the new replicase is always passive in reactions with the original replicase is that it has an alteration in one of its binding sites; the asymmetry in binding scores means that the same string will always be the active and the other passive whenever they have an opportunity to bind.

In this section we consider what will happen if we remove the ability to self replicate from a string, but construct the binding sites such that it is always copied when it encounters the original replicase.

Step by Step Instructions

1. In the 'two molecule' tab:

2. Set strings 1 and 2:

   Original replicase:

   ```
   OOGEOLHHHRLUEUOBBBRBXUUUDYGRHBLROORE$BLUBO^B>C$=?>$$BLUBO%}OYHOB
                    |||   ||||||                                      Differences in binding sites
   ```

   Parasite:

   ```
   OOGEOLHHHRLUEUOBBZZZXUUZZZZZZ
   ```

3. Press ‘New reaction’
4. Press ‘Play’ to run through the whole reaction. Observe that the parasite is copied.

In this reaction we see that the short parasite string is copied. This string is significantly shorter than the replicase molecules we have been considering; consequently the time it takes to complete a copy of this molecule is noticeably shorter.

In the short term, being copied could be considered beneficial. As the parasite is always passive in a reaction with the replicase it does not need the code for replication of a string while the original replicase is in the system. Consider what will happen at the system level when a small population of parasites are added to a ‘bucket’ full of the original replicase

Step by Step Instructions

1. In the ‘Bucket’ tab:
2. Turn off mutation by un-checking the mutation tickbox.
3. Put 300 original replicase molecules into ‘the bucket’.
4. Put 1 parasite molecule into ‘the bucket’.

   Original replicase:

   `OOGEOLHHHRLUEUOBBBRBXUUUDYGRHBLROORE$BLUBO^B>C$=?>$$BLUBO%}OYHOB` Initial number: 300

   Parasite:

   `OOGEOLHHHRLUEUOBBZZZXUUZZZZZZ` Initial number: 1

5. Press the ‘Start’ button.
6. Notice that the original replicase copies itself, as always, but the parasite increases in numbers as time progresses.
7. Notice that after about 10k timesteps, the number of parasites increases dramatically and the number of replicases decreases.
8. The number of parasites peaks at around 600 molecules, by which time the parasite has killed the replicase and so will subsequently die itself.

If you take such a molecule and remove the functional regions, so that it cannot copy itself it becomes a parasite.

The differences in these strings have been hand crafted to give rise to a ‘death spike’ that is approximately 600 molecules high (twice the carrying capacity of the replicase). The blocks of `Z` disable the second binding site. Instead of having a 50:50 chance of being the active molecule in a reaction with the replicase, the parasite is always passive. The length of the molecule is tuned so that simulation does not take too long to run. Much shorter molecules, such as OOGEOL will provide a much higher ‘death spike’, however the real time such a simulation will take is significantly longer than the case we have given.

6 Hypercycles

A ‘hypercycle’ is an interesting phenomena that has been discovered in the evolution of the original replicase. The most basic hypercycle requires two types of molecule. Both of the molecules must be replicase molecules, but must not be self replicases. That is to say, they are able to replicate each other, by two strings of the same kind are not able to make copies of themselves. As you will see in this example, a hypercycle is quite a long way (in evolutionary space) from the original replicase in terms of the direct comparison of strings. They arise in approximately 4% of runs seeded with the original replicase (with mutation turned on).

Step by Step Instructions

1. In the ‘Bucket’ tab:
2. Turn off mutation by unchecking the mutation box.
3. Set up the following configuration:

   Hypercycle 1:

   `OOGEOLHHHRLUEUOBRE$BLUBO^B>C$=?>$$BLUBO%}OYHOB` initial number 50

   Hypercycle 2:

   `BBBRBXUUUDYGRHBLROORE$BLUBO^B>C$=?>$$BLUBO%}OYHOB` initial number 50
4. Press Start.
5. Notice that, unlike the random walk example, the population levels of the two hypercycle molecules are strongly (positively) correlated and stable.
6. Stop the bucket

You may notice that the molecules that make up this hypercycle are constructed from the original replicase. The original replicase has two binding sites, the hypercycle molecules each have one of the two binding sites. In a reaction with each other the alignment of the bind is at the start of both molecules. Consequently, determining which molecule is active and which is passive has a 50:50 chance.

These molecules can also react with themselves, however they do not produce a product. Either molecule reacts with a copy of itself it binds `BLUBO` to `OYHOB`. The instruction pointer simply iterates off the end of the active string terminating the reaction.

You can observe how this works using the ‘two molecule’ tab, either superficially by clicking ‘end’ to see the product of each of the possible reactions; or in depth following each program by stepping through using the ‘next’ button.

7 A mutating bucket

If you have completed all the sections of the worksheet up to this point you will be able to recognise and classify types of populations dynamics in an evolving bucket. You should also have an idea of how to design (or at least edit) stringmol molecules.

1. In the bucket tab.
2. Turn mutation on by putting clicking the check box to show a tick.
3. Start with the original replicase and watch selective sweeps happening.
4. Try seeding the evolving bucket with other molecules - try the hypercycles
5. Would you like to try designing molecules of your own now...