ACS SyntheticBiology

Research Article

pubs.acs.org/synthbio

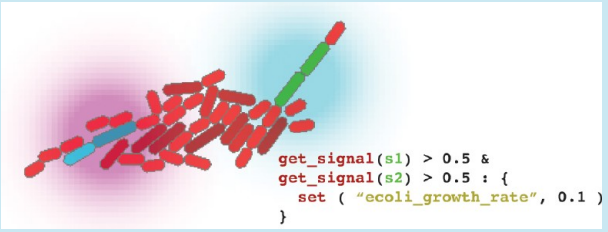
Specification and Simulation of Synthetic Multicelled Behaviors

Seunghee S. Jang, Kevin T. Oishi, Robert G. Egbert, and Eric Klavins*

Department of Electrical Engineering, University of Washington, Seattle, Washington 98195, United States

Supporting Information

ABSTRACT: Recent advances in the design and construction of synthetic multicelled systems in *E. coli* and *S. cerevisiae* suggest that it may be possible to implement sophisticated distributed algorithms with these relatively simple organisms. However, existing design frameworks for synthetic biology do not account for the unique morphologies of growing microcolonies, the interaction of gene circuits with the spatial diffusion of molecular signals, or the relationship between multicelled systems and parallel algorithms. Here, we introduce a framework for the specification and simulation of multicelled behaviors that combines a simple simulation of microcolony growth and molecular signaling with a new specification language called `gro`. The framework allows the researcher to explore the collective behaviors induced by high level descriptions of individual cell behaviors. We describe example specifications of previously published systems and introduce two novel specifications: microcolony edge detection and programmed microcolony morphogenesis. Finally, we illustrate through example how specifications written in `gro` can be refined to include increasing levels of detail about their bimolecular implementations.



KEYWORDS: *multicelled behavior, pattern formation, bacterial growth, specification*

The ability to engineer complex, synthetic, multicelled systems could revolutionize tissue engineering, biomass production, biosensing, and biodetection. Recent advances in this area involve combining small genetic networks in *E. coli* or *S. cerevisiae* with cell-to-cell signaling to produce synthetic coupled oscillators,[1] multicell logic,[2,3] pattern formation,[4−6] and population control.[7] Although these examples are simple compared to the multicelled behaviors found in nature, they do begin to explore some of the basic mechanisms that synthetic biologists must harness to make further progress: environmental response,[8] signaling,[9] genetic network design,[10] and control of growth and apoptosis.[7] To advance the field, more work is needed in understanding and repurposing basic molecular and cellular mechanisms, and the algorithmic foundations of multicelled systems need to be developed to guide how new mechanisms can be used, in what kinds of algorithms, and with what limitations.

Design tools developed for synthetic biology are generally focused on the behavior of single genetic circuits. They model systems using chemical reactions, enzyme kinetics, and gene network models.[11−13] The interactions induced by signaling and their effects on geometry have been examined in, for example, spatial simulations of cell signaling,[14] 3D cell growth,[15] cell networks,[16] and morphogenesis,[17,18] which combine reaction diffusion models[19] with geometrical models of cell growth and division. Typically such simulations are geared toward modeling existing systems and not toward designing new ones. Separately, research in computer networks, distributed systems,[20] multirobot systems,[21] and stochastic self-organization[22] has become quite advanced. In particular, the theoretical foundations for distributed systems[20] and parallel algorithms[23] have enabled new algorithms and also described the inherent limitations[24,25] of distributed computation. Most of the approaches in the distributed systems literature, however, rely on various assumptions that do not hold in biological, multicelled systems: that nodes have unique identifiers, that they can send and receive arbitrary kinds of data, that they can store data in buffers, and so on. Even work *in silico* on pattern formation that uses diffusing signals[26] is difficult to imagine being implemented biologically.

Here we introduce a new tool, called `gro`, intended to assist in the specification, design, and exploration of ideas for multicelled behaviors in a 2D environment where the spatial effects of cell growth, cell crowding, and signal diffusion dominate. `gro` is an open source software package that combines a distributed systems and parallel computing approach with the simulation of up to a few thousand bacterial cells growing in a 2D environment. The simulation component is focused on *E. coli*-like bacterial microcolonies growing in a single layer as would be viewed with a fluorescence microscope.[27] Although only around 10 generations can be grown in this setting before crowding overwhelms the system, we propose that controlling what happens in the initial stages of microcolony formation is an important engineering challenge that could lead to mechanisms for producing synthetic multicelled systems. Our lab and several others are currently

Special Issue: Bio-Design Automation

Received: April 16, 2012
Published: July 23, 2012

   dx.doi.org/10.1021/sb300034m | *ACS Synth. Biol.* 2012, 1, 365−374

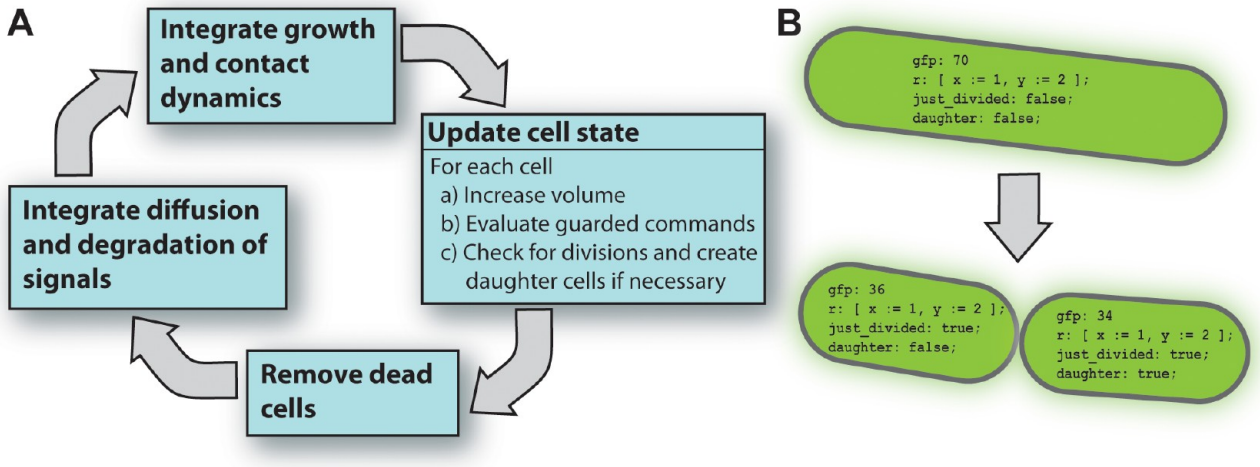


Figure 1. (A) The `gro` main loop. In each iteration of the `gro` main loop, the states of the cells are updated and cell divisions are processed, then cell deaths are accounted for, the diffusion and degradation dynamics of signals are numerically integrated, and finally the cell-to-cell forces due to growth are numerically integrated. (B) Cell division semantics. When a cell divides, the values of the variables of the parent and daughter cells are reassigned. Numerical variables have their states approximately divided in two while variables of other types remain fixed during division. The Boolean variables `just_divided` and `daughter` are assigned just after division for programs to detect the division event.

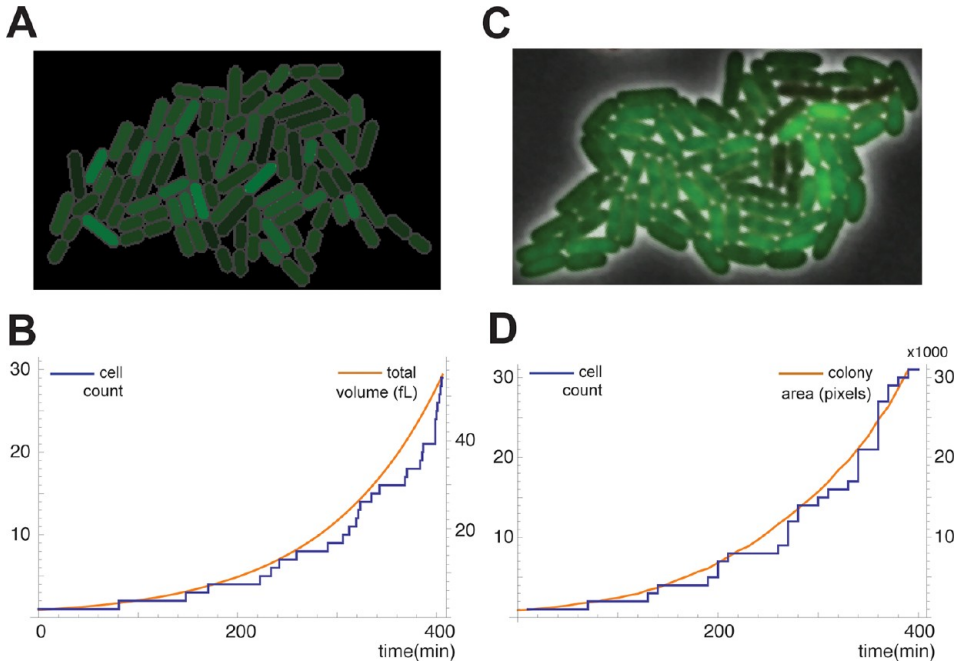


Figure 2. The underlying physics of the `gro` simulator mimic the growth of bacteria in a single layer. (A) A snapshot of the simulator showing a small microcolony. As the cells grow and divide, they push each other away. Also illustrated is the stochasticity of the copy number of a fluorescent marker. (B) Microcolony growth. The volume and the number of cells change over time. Since cell division sizes are random, the number of cells increases stochastically, while the volume increases more smoothly. (C and D) The same data as in panels A and B, but for actual *E. coli* expressing GFP and growing under a fluorescence microscope sandwiched between glass and agar. The growth rate in panel B is set to 0.0081 fL/min, which was obtained by fitting the cell count data in panel D to an exponential growth model. Note that, due to the time-resolution of sampling, not all cell divisions appear as they are in panel B.

working to implement a variety of distributed algorithms in exactly this arrangement.

A key difficulty in such settings is how to understand the mapping between the local behavior of individual cells and the global behavior of a collection of cells. To address this difficulty, the `gro` framework is mainly concerned with allowing researchers to quickly assess whether a scheme that describes what cells should do locally behaves as expected globally. The language is flexible enough to allow systems to be specified at varying levels of detail.

Here, we describe the `gro` simulation environment, which accounts for cell growth, division, noise, crowding and cell−cell forces, and the production, diffusion, and sensing of molecular signals. We compare output from the simulation with experimental data. We introduce the `gro` language, describing its main features and giving several examples. We illustrate how `gro` can be used to specify systems via several examples. First,

dx.doi.org/10.1021/sb300034m | *ACS Synth. Biol.* 2012, 1, 365−374

to ground the user in a familiar example, we describe a simple specification of transcription and translation in a growing microcolony. Second, we introduce a novel specification of a microcolony edge detection behavior that uses stochasticity and diffusing molecular signals. Third, we describe a novel specification of a class of developmental programs that use finite state machines to determine microcolony morphology. Finally, we describe how gro can be used to specify a system at several levels of abstraction, by specifying and simulating a bacterial population control system previously described in the literature[7] using an informal refinement process to increase the fidelity of the specification.

The examples presented in this paper are intended to illustrate the main features of the gro framework. They represent a growing list of examples that we and various initial users of gro at our institution and elsewhere have devised. Many more examples ranging from the specification of mixed consortia of cells to bacterial chemotaxis are included with the online distribution of the gro software.

RESULTS AND DISCUSSION

Simulation. gro provides a realistic simulation of bacterial microcolony formation that is intended to mimic the behavior of *E. coli* cells when they are grown as a monolayer and viewed under a microscope. In separate but integrated processes, gro models growth, division, contact forces between the cells, and small molecule diffusion, as described in Figure 1A. The result is similar to actual bacterial growth as observed in the lab and illustrated in Figure 2.

Cells are assumed to be approximately cylindrical, with radius r = 0.5 μm and an initial length of l = 2.0 μm. The time resolution of each subprocess of the simulation is controlled by the time step parameter *dt*, which can be adjusted in gro programs to reduce inaccuracies due to numerical errors. The volume of a cell in gro is initially $V = \pi r^2 \cdot l$ = 1.57 fL. Its growth rate, k, can be varied. The default value of 0.0081 fL/min results in a doubling time of 85 min, which is close to what we observed in the microscope, as shown in Figure 2. Growth in each cell is modeled according to the differential equation

$$\frac{dV}{dt} = kV$$

which is numerically integrated by gro, using the time-step *dt*, in parallel with the simulation of other subprocesses described below. The volume of a microcolony thus grows exponentially over time, as illustrated in Figure 2B. The volume change qualitatively matches actual *E. coli* growth shown in Figure 2D.

Each cell grows until it has approximately doubled in size, at which point it divides approximately in two (Figure 1B). The mean μ and variance σ^2 of the division size are also parameters that can be set in gro. By default, μ = 3.14 fL and σ^2 = 0.005 fL^2. gro simulates a simple N step counting process tuned to result in the desired division size statistics. Thus, although the volume of a microcolony grows smoothly, the number of the cells increases according to a discrete stochastic process, as illustrated in Figure 2B and compared to actual *E. coli* growth shown in Figure 2D.

Because cells are constrained to a single layer, the physics of their growth and contact forces can be modeled using a simple 2D physics engine.[28] No attempt was made to estimate actual contact forces in real microcolony growth, and the effect is intended to be only qualitatively similar to the actual process. In particular, the cells are modeled as 2D polygons; friction between the cells and the environment is set high enough that the dynamics of their motion is essentially first order; contact forces between the cells are set so that they slide, slightly, relative to each other and produce the characteristic crowding geometry seen in real microcolony growth. In each step of the main loop of the gro simulation, the volumes (and therefore the lengths) of the cells are adjusted according to the above, and then the physics engine is called to simulate the resulting contact forces and produce motion for *dt* simulated minutes.

Cell to cell communication via small molecules is simulated using the finite element method with a 2D grid of square elements, the resolution of which can be specified in gro. The dynamics are simulated using Euler integration, with the same time step of *dt* minutes as is used in growth and motion. When a new signaling molecule is declared, its diffusion and degradation rates can be specified. Cells can emit, sense, and absorb small molecules. For example, two cells in the simulation in Figure 5B are emitting different signals, and the cells between them are programmed to grow only when they receive both signals, which incidentally always results in a microcolony with the blue cell on one side and the green cell on the other (see below for details).

In addition to *E. coli* cells, gro provides an interface for developers to define new types of cells using whatever geometry and physics they desire. The level of detail and realism of such simulations is left to the discretion of the programmer. Finally, gro simulates a chemostat mode (see Figure 6), reminiscent of that used by Hasty's group for their work on coupled oscillators,[1] in which cells grow in a chamber past the bottom of which nutrients flow at a high enough velocity that cells are swept away if they move too close. The population control system described below uses the chemostat mode.

Guarded-Command Programming. Each cell in the simulation runs a program written in the gro programming language. The language is designed to allow behaviors to be expressed at whatever is the most useful level of abstraction for the current design phase. For example, a program written at a high level of abstraction might state that a cell should emit a signaling molecule at a certain rate. On the other hand, a program encoding a more detailed description of the same process might state that a certain gene should be stochastically transcribed and translated to produce an enzyme that converts a precursor metabolite into a signaling molecule that diffuses through the cell membrane at a rate proportional to the difference between the intracellular and extracellular concentrations. For example, the system on population control described below and in Figures 6 and 7 can be specified at three different levels of abstraction, depending on what details are of interest.

gro is a strongly typed, interpreted programming language. Variable types include integers, floats, Booleans, strings, lists, records, lambda expressions, and recursive functions. gro can be easily interfaced with other programming languages and via this mechanism provides function libraries for math and I/O, for example. Most of the details of gro are standard and are described in the online reference manual. Only what makes gro unique and especially relevant for the specification of highly parallel and stochastic behaviors in multicelled systems is described here.

To model parallelism, gro programs consist of sets of unordered *guarded commands*[29,30] of the form *g*:*c* where *g*, the guard, is a Boolean expression and *c*, the command, is a list of

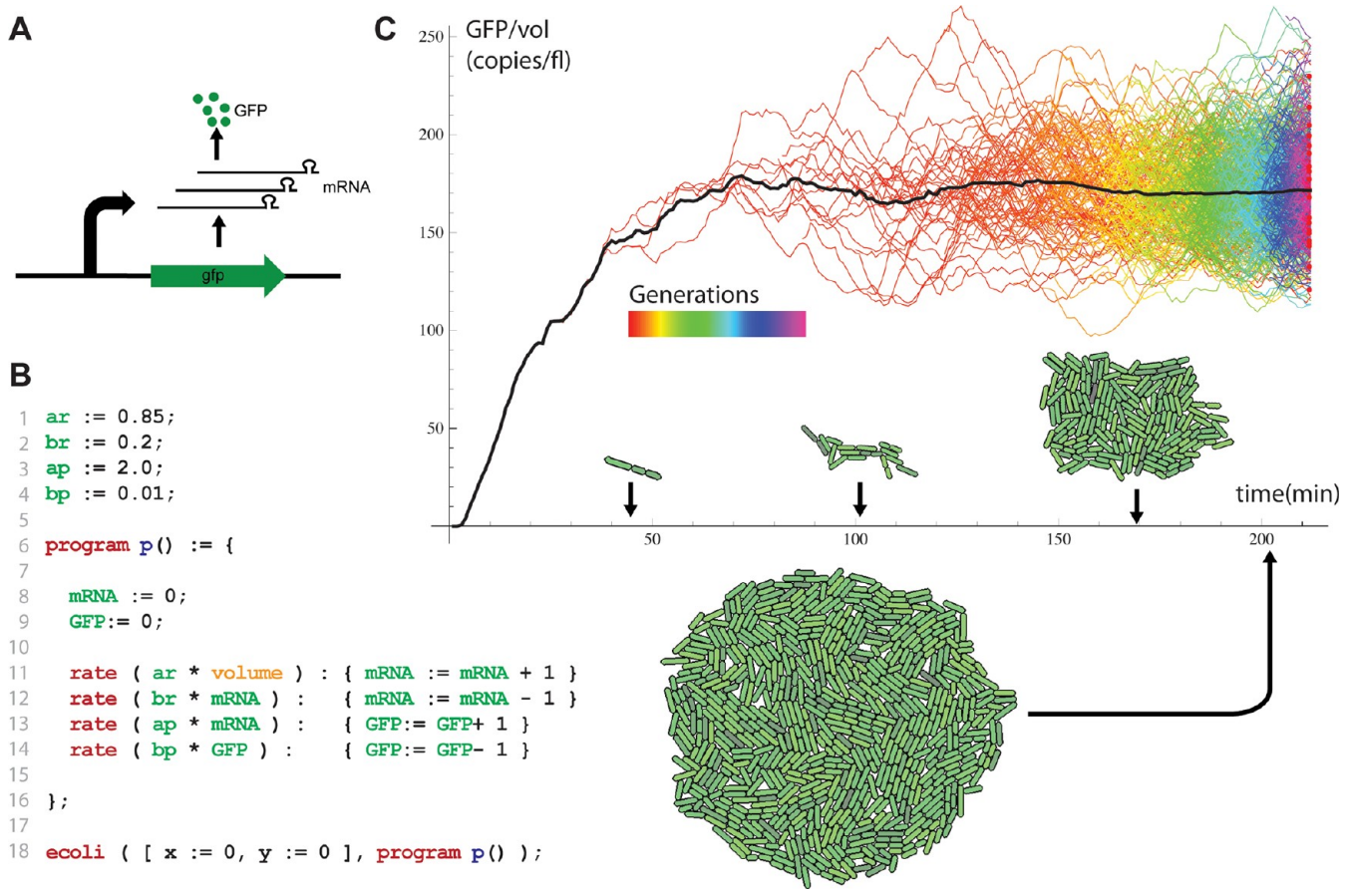


Figure 3. Specification of GFP production in a growing microcolony of *E. coli*. (A) GFP production in which a single gene is transcribed to produce mRNA, which is translated to produce the GFP protein. (B) A `gro` program encoding the model. Physical constants are declared first. Then the program `p()` is defined and includes two variable initializations and four guarded commands that encode production and degradation reactions. Finally, the program is associated with a single cell. (C) The initial cell grows and divides. The copy numbers of the mRNA and GFP molecules change over time due to stochastic gene expression and stochasticity in cell division. Each different colored trace represents the GFP/volume of a different single cell. The black line represents the average of the GFP/volume taken over all the cells.

statements that can be either assignments or function calls. For several examples, see the `gro` programs in Figures 3B, 4B, and 6B, which are explained in more detail below. The meaning of a guarded command is as follows: in each step of the simulation, each guard is evaluated. If it evaluates to true, then the associated command is evaluated. The intention is that each guarded command specifies a distinct process in the cell and that all such processes occur effectively simultaneously. Following a standard approach in modeling parallelism,[23] guarded commands are executed in an unspecified order despite being listed in a particular order on the page.

To model stochastic events in the cell, `gro` provides a special function `rate` that takes one argument and returns true or false randomly. In particular, the expression `rate (r)` returns true upon a given evaluation with probability $r \cdot dt$ and false the rest of the time, where dt is the simulation time step. Thus, the rate function allows `gro` programs to approximate the Master Equation with Euler integration (as in, for example, Figure 3). The use of Gillespie's *Stochastic Simulation Algorithm*[31] is unfortunately not practical due to the necessity to concurrently simulate growth, geometry, and signal dynamics.

Note that any chemical reaction network or gene network model has a trivial mapping to a guarded command program. Thus, `gro` specifications form a (strict) superset of master equation models, which are reasonably well accepted as models of the parellel execution of chemical reactions in the cell. At the same time, since `gro` allows for arbitrary expressions in guards and commands, considerably more concise specifications of behavior than could be obtained with sets of chemical reactions can be built.

In detail, the interpretation of a `gro` program proceeds as follows. After a `gro` file is parsed and type-checked, the programs associated with cells are initialized, meaning that any commands outside of guards are executed. The initial locations and orientations of cells are also initialized at this time. Next, as illustrated in Figure 1A, the simulation repeatedly performs the following steps:

- The volume and guarded command program of each cell is updated, and division conditions are checked. If the cell divides, the variables in the mother and daughter cell are adjusted, and a copy of the program and its internal state are associated with the daughter cell.
- Dead cells are removed.
- The concentrations of signaling molecules are updated according to a numerically integrated reaction/diffusion process.
- Growth dynamics and contact forces are numerically integrated by the underlying 2D physics engine.

dx.doi.org/10.1021/sb300034m | *ACS Synth. Biol.* 2012, 1, 365–374

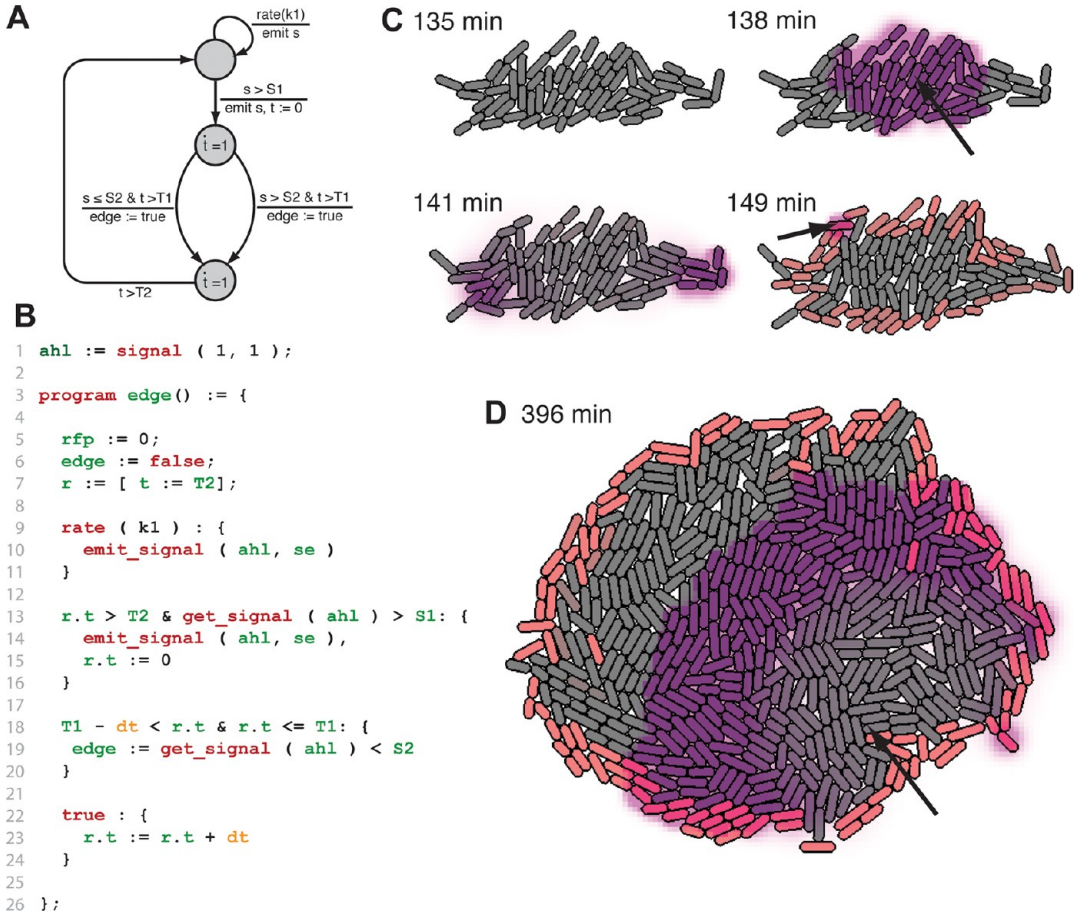


Figure 4. Specification of an edge detection behavior. (A) A timed automaton specification of the edge detection system. (B) The edge() program in `gro`. Cells either stochastically start waves or propagate waves. If the concentration of the signal is below a the threshold some number of minutes after propagating a wave, the cell is likely near the edge of the microcolony. (C) The first wave, initiated in the middle of a microcolony, propagates to the edge. Cells detecting the edge condition turn red. (D) A more mature microcolony in which cells have already detected the edge. Spontaneous wave behavior continues, reinforcing edge detection.

A cell divides when a random condition set with the cell-specific `division_size_mean` and `division_size_variance` parameters, described above, is met. When division occurs, the volume of the parent cell is cut approximately in half. The program associated with the cell is copied and associated with the daughter cell. Importantly, values of all numeric local variables in the programs are cut approximately in half and apportioned to the mother and daughter cells as well, modeling the discrete noise induced by division on copy number,[32] as shown in Figure 1B. Note that to *protect* variables from being cut in half upon division, `gro` programs often hide numeric variables inside records (as with, for example, `r.t` in Figure 4A), allowing specifications to state that cells must pass state information unchanged to their progeny.

Examples. Programs can affect their host cell's behavior in a variety of ways. As mentioned above, they can set the growth rate of the cell and can cause the cell to emit or sense signals. In addition, programs can force the cell to divide (no matter what its size) or to undergo apoptosis (Figure 6), and of course, because `gro` allows one to write arbitrarily complex programs, any internal logic, dynamics, memory, data structures, or stochastic processes can be constructed.

We illustrate the expressivity of `gro` with three examples. First, to ground the reader in a familiar example, we illustrate how `gro` can be used to specify transcription and translation at a high level. Second, we introduce a novel microcolony edge detection behavior. Third, we describe a novel approach to determining microcolony morphology. The complete code listings and videos of the simulations are included in the Supporting Information.

Figure 3 illustrates how `gro` might be used to build a simple model of the transcription and translation of a protein, here GFP, in *E. coli* as shown in Figure 3A. It is common, for example, to write a Master Equation describing transcription and translation as well as the degradation and dilution of mRNA and protein.[33] Although the ability of `gro` to predict microcolony shape and cell−cell variability is not strictly necessary in this case, we believe the example nicely illustrates how standard models of gene expression are easily expressed in the guarded-command formalism. The `gro` program in Figure 3B encodes the reactions. In lines 4−10, parameters for production and degradation are defined. Next, a program `p()` is defined with two variable declaration/instantiations (lines 8 and 9) and four guarded commands (lines 11−14). The guarded commands define the rates and effects of transcription, mRNA degradation, translation, and protein degradation, respectively. Note that because the volume of the cells running this program will be increasing, dilution is not specified directly by the program but is a side effect of the cell's increase in

volume over time. The last line of the program associates the program with a single simulated *E. coli* cell at the center of the simulated environment. Running this program results in a growing microcolony of cells as shown in Figure 3C. Initially there is one cell. By 200 min there are hundreds of cells. Note that `gro` interprets the variable `gfp` as green fluorescent protein and renders the cell with which it is associated in green using an intensity proportional to the concentration of GFP in that cell. As can be seen in the figure, the amount of GFP molecules per unit volume fluctuates for a given cell; is initially correlated with the parent cell after division and becomes uncorrelated over time; and stabilizes in mean and variance over time.

Figure 4 illustrates that, through signaling, cells in a microcolony can sense some aspects of the geometry of their environment. In particular, we show that cells can detect whether or not they are near the edge of the growing microcolony. A *timed automaton*[34] that specifies a novel edge detection behavior is shown in Figure 4A, and its translation into a `gro` program is shown in Figure 4B. The system behaves as follows. With low probability, an arbitrary cell emits a pulse of a signaling molecule, here suggestively called ahl (lines 9–11). Cells that sense that their local concentration of ahl has increased beyond a threshold emit a pulse as well and then enter a refractory period where they do not emit signals (lines 13–16). The result is a wave of increased signal concentration that travels from within the microcolony outward toward the edge. To detect whether it is close to an edge, a cell can use the fact that neighboring cells, if there are any, will signal immediately after it does; so if the signal level is below some threshold value immediately after a signal is sent, the cell must be near the edge (lines 18–20). Cells that have sensed they are near the edge produce RFP (code not shown), while other cells actively degrade it, acting as a simple low pass filter of the Boolean edge condition. The behavior is illustrated in Figure 4C, which shows the random initiation of the first pulse by one of the cells and the effect on edge cells. Figure 4D shows the microcolony sometime later, after many such pulses.

Figure 5A describes a specification of a developmental program, encoded in `gro`, that produces microcolonies with a specific structure: three bands of cells each expressing a different reporter. The specification is given by a finite state machine (the `gro` code for which is left to the Supporting Information). Each state of the program specifies a growth rate, a reporter protein, and whether to emit a given signal. Transitions occur either upon cell division, in which case they are mother-daughter[35] specific, or after timers have elapsed. In state 1, the cell grows at a rate of 0.01 fL/min. When it divides, one of the cells (arbitrarily called the mother cell[36]) continues in state 2, while the other cell continues with a copy of the state machine in state 3. In states 2 and 3, the cells emit cyan and magenta signals, respectively, and continue to grow. In addition, when a cell in state 2 (or 3) divides, one of the resulting cells continues in state 2 (or 3) and the other continues in state 4. Cells in state 4 express RFP, and their growth rates are high only if they receive both the cyan and magenta signals (denoted by the ∧ symbol). This condition allows only RFP cells that are located between the cells in states 2 and 3 to grow, and it pushes the signal-emitting cells in states 1 and 2 apart. Finally, after a specified amount of time (T_1), cells in states 1 and 2 switch to states 5 and 6, where they continue to grow, do not emit signals, and express the state-specific reporters GFP and CFP, respectively. After another

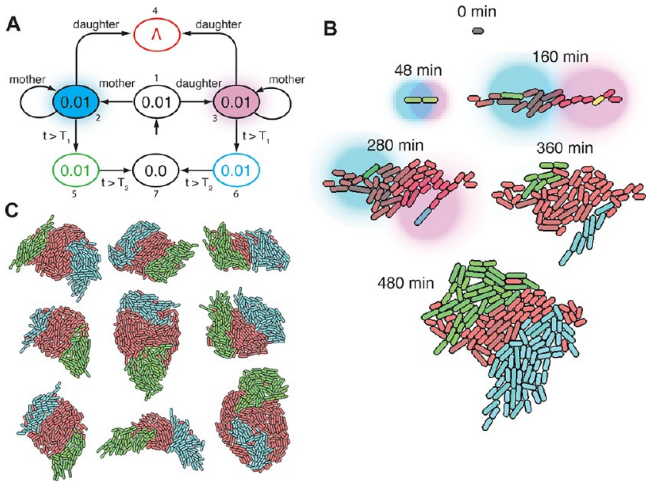


Figure 5. Programmed morphology. (A) An example finite state machine that guides the behavior of a given cell to produce bands. Cells start in the center state and switch to new states based on whether they are mother cells or daughter cells after division or if timers have gone off. (B) An example run of the specification in panel A. Starting with a single cell, the cells divide and take on different roles eventually forming the three desired bands of fluorescent protein expressing cells. (C) Several example runs, highlighting the stochastic nature of the system's behavior.

timer elapses, cells in states 5 and 6 switch to state 7, where all activity stops. Because the ancestors of the GFP and CFP cells were pushed apart by the RFP cells, the GFP and CFP cells grow on the ends of the microcolony, resulting in the desired three-band pattern. One run of the program is shown in Figure 5B, and Figure 5C shows the results of nine more runs, demonstrating that the three-band structure is approximately achieved in each case, but with some variation due to the stochastic effects of cell division and growth.

Refinement. An advantage of using a first-class programming language over, for example, chemical reactions to specify cell behavior is that it allows systems to be specified at whatever level of abstraction is convenient. To illustrate this point, we examine several increasingly refined specifications of the population control circuit in Figures 6A and 7A, devised by You et al.,[7] in which a gene that induces cell-death (*ccdB*) was placed under the control of a quorum sensing promoter. In the original system, the LuxI enzyme synthesizes the signaling molecule AHL, and the transcription factor LuxR acts as an activator when bound by AHL. As the cell density increases, the concentration of AHL also increases, and the downstream promoter is activated, resulting in increased apoptosis, leading to lower AHL concentration. The result is a negative feedback loop that stabilizes the population at a low density.

A simple and elegant specification of this behavior is shown in Figure 6B. Lines 5–7 cause the cells to continually emit a signal, which diffuses through the environment, shown in cyan in Figure 6D. Lines 9–11 cause the cells to lyse probabilistically, at a rate proportional to the concentration of the signal. The specification is parametrized by the diffusion rate of the signal (k_1), the degradation rate of the signal (k_2), the rate at which the signal is emitted (k_3), and the rate constant for lysis (k_4). Figure 6C and D shows the effect of varying, for example, k_2 as was done by adjusting the acidity of the media in the original paper.

dx.doi.org/10.1021/sb300034m | *ACS Synth. Biol.* 2012, 1, 365–374

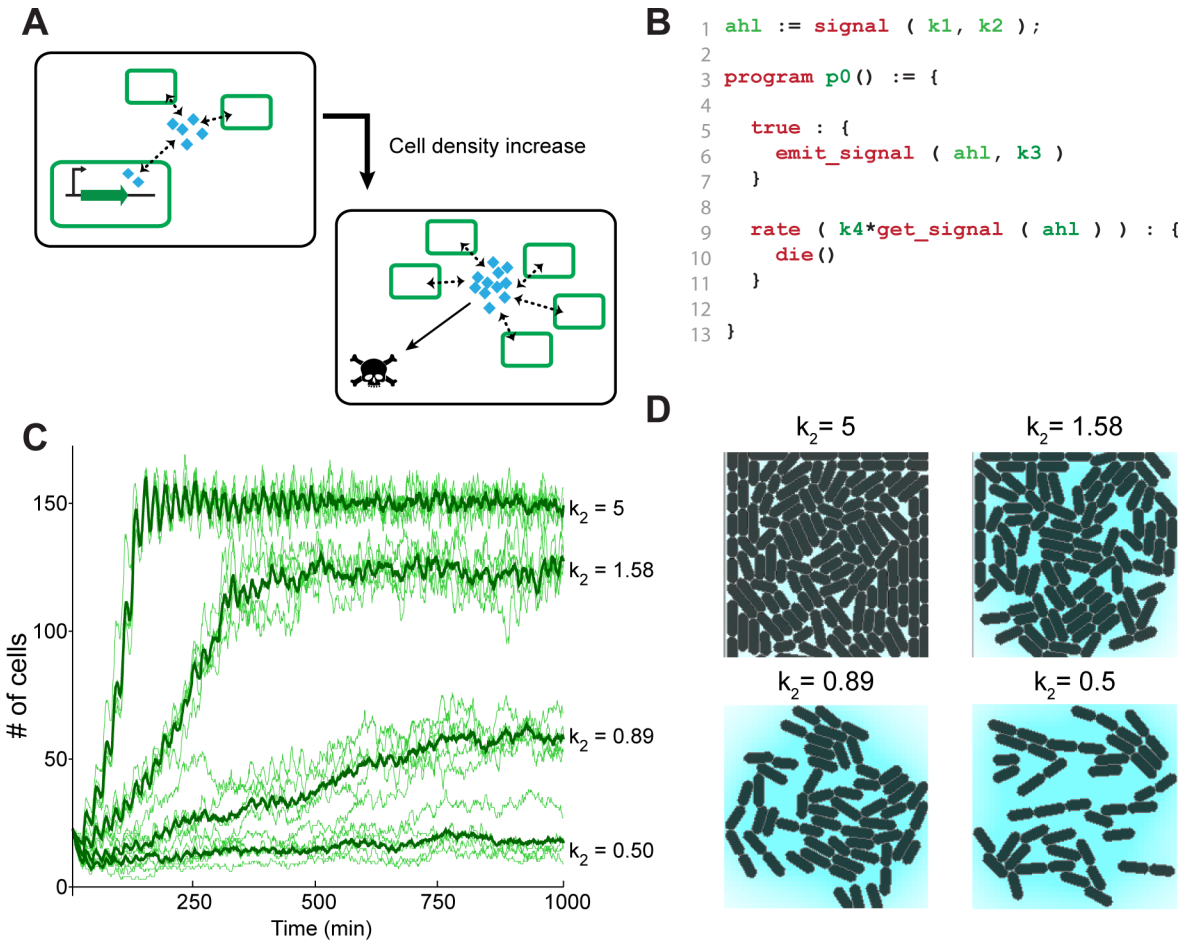


Figure 6. High-level specification of the population control circuit introduced by You et al.[7] (A, B) Cells emit a signal, which diffuses through the environment. The cells undergo lysis at a rate proportional to concentration of the signal. The parameters k_1–k_4 can be tuned to produce a variety of effects. (C) By varying the degradation rate k_2 of the signal, different population densities can be achieved. The plot shows the number of cells in a fixed volume microchemostat versus time. (D) Snapshots of the population control system running in `gro`'s chemostat mode, using different values of the degradation rate parameter.

A simple specification can be useful for exploring whether an idea is sound without getting into the details of how the idea is implemented. The next step is to fill in those details. Figure 7B shows one such refinement, in which the second guarded command in Figure 6B is replaced by a set of guarded commands that implement the production and degradation of CcdB, as well as making the lysis rate proportional to the level of CcdB, instead of to the signal concentration directly. A further refinement is shown in Figure 7C where the production and sensing of the signal is governed by the *luxR*/*luxI* system. The resulting system behaves similarly to the specification in Figure 6B, as is shown in Figure 7D, which shows, as an example, that the sensitivity of the system's behavior to the rate constant k_2 is maintained, even though the interpretation of the constant is slightly different in each system.

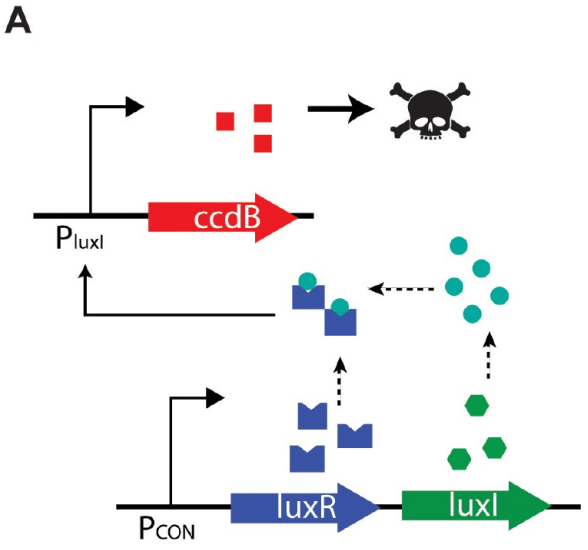
Conclusion. We have described a framework for the simulation of high-level specifications of multicelled behaviors in 2D environments with diffusing molecular signals. Our main goal is to provide a framework in which the global geometrical behavior resulting from a specification of the local behaviors of cells can be explored. We have focused on examples in which geometry and signaling are important, such as with the microcolony edge detection and morphogenesis examples. A host of other simulation and modeling environments are readily available for exploring bulk behaviors in well mixed environments where `gro` may not be appropriate.

Central to our approach is the concept of abstraction: a system's behavior and how it is implemented can be treated separately. This observation has been made in, for example, models of osmo-adaptation[37] in yeast in which a simple second order system with a few parameters abstracts away tens of biochemical reactions that actually implement the reaction in the cell. Similarly, specifications of behaviors made during the design phase should capture the overall functionality of a system allowing the engineer to test whether the logic of a communication scheme is sound and produces the desired results before work implementing the low-level details is begun.

For example, in designing the edge detection example we did not start with gene networks or biochemical reactions, we started with a simple idea: a random event triggers a burst of a signal molecule that is propagated through the microcolony. Coupled with the detection of the signal concentration after a delay, this basic capability produces the desired effect. Similarly, in the developmental example, we specified the system using a finite state machine to test whether the logic it specifies actually would reliably produce banded microcolonies.

The microcolony edge detection and the microcolony morphogenesis example specifications are written at a high level so that whether they produce the desired global behavior

dx.doi.org/10.1021/sb300034m | *ACS Synth. Biol.* 2012, 1, 365–374



B

```
program p1() := {

  CcdB := 0;

  true : { emit_signal ( ahl, k3 ) }

  rate ( k5*get_signal ( ahl ) ) : {
    CcdB := CcdB + 1
  }

  rate ( k6 * CcdB ) : { CcdB := CcdB - 1 }

  rate ( k4 * CcdB ) : { die() }

}
```

C

```
program p2() := {

  CcdB := 0;
  LuxI := 0;
  LuxR := 0;

  true : { emit_signal ( ahl, k3 * LuxI ) }

  rate ( k5*get_signal ( ahl ) * LuxR ) : {
    CcdB := CcdB + 1;
  }

  rate ( k6 * CcdB ) : { CcdB := CcdB - 1 }

  rate ( k7 * volume ) : { LuxI := LuxI + 1 }

  rate ( k8 * LuxI ) : { LuxI := LuxI - 1 }

  rate ( k9 * volume ) : { LuxR := LuxR + 1 }

  rate ( k10 * LuxR ) : { LuxR := LuxR - 1 }

  rate ( k4 * CcdB ) : { die() }

}
```

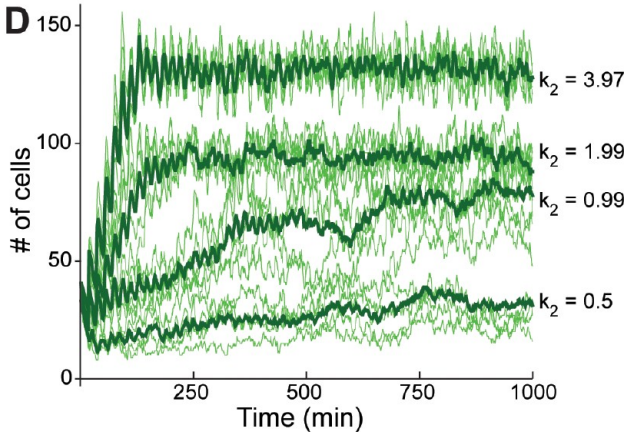


Figure 7. Two refinements of the population control specification `p0` described in Figure 5. (A) A model of autonomous population control involving three proteins: LuxI synthesizes AHL, the LuxR transcription factor is activated by AHL and activates pLuxI promoter, and CcdB triggers cell apoptosis. (B) The section of the original specification p0 that causes apoptosis has been replaced by reactions describing production and degradation of the protein CcdB. The rate of lysis has been changed to be proportional to the amount of the CcdB protein. (C) A further refinement in which the production and sensing of the signal are controlled by the LuxI and LuxR proteins, respectively. New levels of detail require more parameters, in this case k_5–k_{10}. (D) The refinements produce, qualitatively, the same behavior as does the original specification, as shown in this plot of number of cells (in a fixed volume microchemostat) versus time for different values of k_2.

can quickly be assessed through simulation. How these specifications might be implemented using existing signaling molecules, enzymes, and/or transcription factors is of course a difficult and open question beyond the scope of this paper. However, we believe that many of the prerequisites have been demonstrated in the literature: noise generation,[38] concentration detection and filtering,[4,39] state control,[40,41] Boolean logic,[2] and signaling.[1] In fact, the composition of these mechanisms into more complex systems is the subject of intense research in the synthetic biology community.

It is interesting to note that most specifications reuse a few basic ideas: that cell–cell signaling can be tuned; that cells can control their growth rates; that cells can undergo apoptosis; that signals can be filtered and thresholded; that randomness can be harnessed; that asymmetries after cell division can be amplified; and that the passage of time can to some extent be monitored. By combining such low-level capabilities in new ways, a variety of behaviors may be obtainable. To the extent that `gro` becomes a useful design tool, it may be used to simultaneously guide the exploration of reliable implementations of the above primitives as well as their composition into increasingly complex programs that guide construction of novel multicelled behaviors.

METHODS

The `gro` language is based on CCL[30] and is implemented in lex, yacc, and C++. The rigid body dynamics and growth of cells is implemented using the chipmunk[28] 2D physics library of functions, which includes primitives for 2D polynomials, contact forces, and friction. Signals are implemented with a custom finite element simulation, with a grid size and extent that can be controlled from the `gro` language. The graphics in

the simulation and the user interface is implemented with Qt. The gro system has been compiled for Mac OS X and Windows and can be obtained online. Data is output from gro to standard output and piped into comma separated value files that are then read by *Mathematica* or similar software for analysis and presentation.

E. coli strain MG1655 was transformed with a mini-F plasmid encoding the $lacI^q$ allele, and a high copy plasmid (colE1 origin) encoding *gfp* under the control of a synthetic promoter with dual *lac* operator sites (BioBrick R0011). Individual colonies were grown at 37 °C to saturation in M9CA glucose media (M8010, Teknova) supplemented with 34 μg/mL chloramphenicol and 100 μg/mL ampicillin. The culture was then diluted 200:1 in the same media and incubated for 1 h at 37 °C. A small volume of the culture was added to a glass-bottomed microscopy dish (GWSB-3512, WillCo Wells BV), and the cells were immobilized to the glass surface by a 1% nutrient agar slab prepared with M9CA glucose media, 34 μg/mL chloramphenicol and 100 μg/mL ampicillin. Cell samples were then transferred to a Nikon Ti-S inverted microscope (Nikon Instruments) equipped with a 12-bit Coolsnap-EZ camera (Roper Scientific) and an environmental chamber (In Vivo Scientific). Time lapse movies were created by capturing phase contrast images of the growing microcolonies every 10 min, and GFP images every 20 min over a 10 h incubation period at 32 °C. Colony size was determined using Matlab by summing the number of pixels in an area resulting from thresholding the phase contrast images, followed by a morphological closing operation. The number of cells was counted manually.

■ ASSOCIATED CONTENT

Supporting Information

The complete code for the examples in the paper and videos of simulations of the examples. This material is available free of charge via the Internet at http://pubs.acs.org.

■ AUTHOR INFORMATION

Corresponding Author

*E-mail: klavins@uw.edu.

Notes

The authors declare no competing financial interest.

The gro interpreter, simulator, documentation, and many example gro specifications are available at http://depts.washington.edu/soslab/gro/.

■ ACKNOWLEDGMENTS

The authors thank Erik Winfree, David Soloveichik, and Matthew Thomson who used gro in their courses at Caltech and UCSF, respectively. We thank the students at the University of Washington who used gro in our Autumn 2011 course on synthetic biology. In particular Rehana Rodrigues won "best gro specification" with a program on edge detection from which the example in this paper was derived. Finally, we would like to thank the many alpha testers who have downloaded initial versions of gro and provided helpful feedback. This work was funded in part NSF grant number 1137266 and in part by a grant from the Paul G. Allen Family Foundation.

■ REFERENCES

(1) Danino, T., Mondragón-Palomino1, O., Tsimring, L., and Hasty, J. (2010) *Nature 463*, 326−330.

(2) Tamsir, A., Tabor, J., and Voigt, C. (2011) *Nature 469*, 212−215.

(3) Regot, S., Macia, J., Conde, N., Furukawa, K., Kjellen, J., Peeters, T., Hohmann, S., de Nadal, E., Posas, F., and Sole, R. (2011) *Nature 469*, 207−211.

(4) Basu, S., Gerchman, Y., Collins, C., Arnold, F., and Weiss, R. (2005) *Nature 434*, 1130−1134.

(5) Sohka, T., Heins, R., Phelan, R., Greisler, J., Townsend, C., and Ostermeier, M. (2009) *Proc. Natl. Acad. Sci. U.S.A. 106*, 10135−10140.

(6) Liu, C., Fu, X., Liu, L., Ren, X., Chau, C., Li, S., Xiang, L., Zeng, H., Chen, G., Tang, L., Lenz, P., Cui, X., Huang, W., Hwa, T., and Huang1, J. (2011) *Science 334*, 238−241.

(7) You, L., Cox, R., Weiss, R., and Arnold, F. (2004) *Nature 428*, 868−871.

(8) Tabor, J., Salis, H., Simpson, Z., Chevalier, A., Levskaya, A., Marcotte, E., Voigt, C., and AD, A. E. (2009) *Cell 137*, 1272−1281.

(9) Waters, C., and Bassler, B. (2005) *Annu. Rev. Cell Dev. Biol. 21*, 319−346.

(10) Bashor, C., Horwitz, A., Peisajovich, S., and WA, W. L. (2010) *Annu. Rev. Biophys. 9*, 515−537.

(11) Chandran, D., Bergmann, F., and Sauro, H. (2009) *J. Biol. Eng. 3*, 19.

(12) Myers, C., Barker, N., Jones, K., Kuwahara, H., Madsen, C., and Nguyen, N. (2009) *Bioinformatics 1*, 2848−2849.

(13) Hucka, M., et al. (2003) *Bioinformatics 9*, 524−531.

(14) Angermann, B. R., Klauschen, F., Garcia, A. D., Prustel, T., Zhang, F., Germain, R. N., and Meier-Schellersheim, M. (2012) *Nat. Methods 9*, 283−289.

(15) Mogilner, A., and Odde, D. (2011) *Trends Cell Biol. 21*, 692−700.

(16) Montagna, S., and Viroli, M. (2010) *Electron. Notes Theor. Comput. Sci. 268*, 115−129.

(17) Affolter, M., Zeller, R., and Caussinus, E. (2009) *Nat. Rev. Mol. Cell Biol. 10*, 831−842.

(18) Dupuy, L., Mackenzie, J., Rudge, T., and Haseloff, J. (2007) *Ann. Botany 101*, 1255−1265.

(19) Kondo, S., and Miura, T. (2010) *Science 329*, 1616−1620.

(20) Lynch, N. (1996) *Distributed Algorithms*, Morgan Kaufmann, San Francisco.

(21) Michael, N., Fink, J., and Kumar, V. (2011) *Autonomous Robots 30*, 73−86.

(22) Klavins, E. (2007) *Control Syst. Mag. 24*, 43−56.

(23) Chandy, K. M., and Misra, J. (1988) *Parallel Program Design: A Foundation*, Addision-Wesley Reading, MA.

(24) Fischer, M. J., Lynch, N. A., and Paterson, M. S. (1985) *J. ACM 32*, 374−382.

(25) Klavins, E. (2004) in *Algorithmic Foundations of Robotics V* (Boissonnat, J.-D., Burdick, J., Goldberg, K., and Hutchinson, S., Eds.) Vol. 7, pp 275−292, Springer, Berlin/Heidelberg.

(26) Abelson, H., Allen, D., Coore, D., Hanson, C., Homsy, G., T.F. Knight, J., Nagpal, R., Rauch, E., Sussman, G., and Weiss, R. (2000) *Commun. ACM 43*, 74−82.

(27) Young, J., Locke, J., Altinok, A., Rosenfeld, N., Bacarian, T., Swain, S., Mjolsness, E., and Elowitz, M. (2012) *Nat. Protoc. 7*, 80−88.

(28) Chipmunk 2 Physics Library. http://chipmunk-physics.net/.

(29) Dijkstra, E. W. (1975) *SIGPLAN Not. 10*, 2−2.13.

(30) Klavins, E., and Murray, R. M. (2004) *IEEE Pervasive Comput. 3*, 56−65.

(31) Gillespie, D. T. (1977) *J. Phys. Chem. 81*, 2340−2361.

(32) Huh, D., and Paulsson, J. (2011) *Proc. Acad. Natl. Sci. U.S.A. 108*, 15004−15009.

(33) Mettetal, J., Muzzey, D., Pedraza, J., Ozbudak, E., and van Oudenaarden, A. (2006) *Proc. Acad. Natl. Sci. U.S.A. 103*, 7304−7309.

(34) Alur, R., and Dill, D. L. (1994) *Theor. Comput. Sci. 126*, 183−235.

(35) Lindner, A., Madden, R., Demarez, A., Stewart, E., and Taddei, F. (2008) *Proc. Acad. Natl. Sci. U.S.A. 105*, 3076−3081.

(36) Christen, M., Kulasekara, H., Christen, B., Kulasekara, B., Hoffman, L., and Miller, S. (2010) *Science 328*, 1295−1297.
(37) Mettetal, J. T., Muzzey, D., Gomez-Uribe, C., and van Oudenaarden, A. (2008) *Science 319*, 482−484.
(38) Lu, T., Ferry, M., Weiss, R., and Hasty, J. (2008) *Phys. Biol. 5*, 036006.
(39) Kampf, M. M., Engesser, R., Busacker, M., Horner, M., Karlsson, M., Zurbriggen, M. D., Fussenegger, M., Timmer, J., and Weber, W. (2012) *Mol. BioSyst. 8*, 1824−1832.
(40) Gardner, T., Cantor, C., and Collins, J. (2000) *Nature 403*, 339−342.
(41) Bonnet, J., Subsoontorn, P., and Endy, D. (2012) *Proc. Acad. Natl. Sci. U.S.A. 109*, 8884−8889.