Visual DSD:
a design and analysis tool for DNA strand displacement systems

Matthew R. Lakin, Simon Youssef, Filippo Polo, Stephen Emmott and Andrew Phillips
Microsoft Research, Cambridge

(ppt by John Reif)


• Visual DSD (DNA Strand Displacement) software tool:

- allows rapid prototyping and analysis of computational devices implemented using DNA strand displacement web-based graphical interface.


DSD Provides:
- stochastic and deterministic simulation,
- construction of continuous-time Markov chains
- various export formats (allowing models to be analyzed using third-party tools)
Example:
Toehold-mediated DNA branch migration and strand displacement.
<table>
<thead>
<tr>
<th>strand</th>
<th>syntax</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>⟨S⟩</td>
<td>upper strand with sequence S</td>
</tr>
<tr>
<td></td>
<td>S</td>
<td></td>
</tr>
<tr>
<td></td>
<td>{S}</td>
<td>lower strand with sequence S</td>
</tr>
<tr>
<td></td>
<td>S</td>
<td></td>
</tr>
<tr>
<td>G</td>
<td>{L'}⟨L⟩[S⟩⟨R⟩⟩{R'}</td>
<td>double stranded complex [S] with overhanging single strands ⟨L⟩, ⟨R⟩ and {L'}, {R'}</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>G1 : G2</td>
<td>gates joined along a lower strand</td>
</tr>
<tr>
<td></td>
<td>G1 : : G2</td>
<td>gates joined along an upper strand</td>
</tr>
<tr>
<td>D</td>
<td>A</td>
<td>strand A</td>
</tr>
<tr>
<td></td>
<td>G</td>
<td>gate G</td>
</tr>
<tr>
<td></td>
<td>D1</td>
<td>D2</td>
</tr>
<tr>
<td></td>
<td>new N D</td>
<td>system D with private domain N</td>
</tr>
<tr>
<td></td>
<td>X (n)</td>
<td>module X with parameters (n)</td>
</tr>
</tbody>
</table>

Syntax of the DNA strand displacement (DSD) language, in terms of strands A, gates G and systems D.
Where present, the graphical representation below is equivalent to the program code.
signal transducer gates in series. The first gate should turn a reactive species are those in which all toeholds occur in the input signal and pair of transducers are given by:

$$S(1,x_0) \mid T(1,x_0,x_1) \mid T(1,x_1,x_2)$$

The first ejects an intermediate strand from the input leaving only unreactive waste. We say that a strand or strands_reactive present in the system to cause a strand to be displaced, which the execution of the gates has completed successfully. This is performed with the following code, which the execution of the gate so far), whereas the second design (when we move on to more complicated gate designs. Note, however, that the definition of A correct gate design is closely related to the two-domain design does not include a similar 'new' declaration for the 'a' domain. This has implications for crosstalk between gate populations, as we shall see below. A correct gate design is closely related to the two-domain design, which is, in a generic form, designed to be applicable to various different designs:

![Diagram]

Here, we will focus on verifying the correctness of two examples of initial transducer gate:

(a) Initial species and (b) expected final species for the transducer gate.

Example of initial transducer gate:
(a) Initial species and (b) expected final species for the transducer gate.
Example of initial transducer gate code, with additional definition for signal strands.
Model Checking
Is an automated formal verification technique, based on the exhaustive construction and analysis of a finite-state model of the system being verified.
- The model is usually a labelled state-transition system, in which each state represents a possible configuration of the system and each transition between states represents a possible evolution from one configuration to another.
- The desired correctness properties of the system are typically expressed in temporal logics, such as computation tree logic (CTL) or linear-time temporal logic.

Example typical CTL formulae (along with their corresponding informal meanings):
— A [ G !(“access1” & “access2”): ‘processes 1 and 2 never simultaneously access a shared resource’
— A [ F “end” ]: ‘the algorithm always eventually terminates’
— E [ !“fail” U “end” ]: ‘it is possible for the algorithm to terminate without any failures occurring’.

Model Checker:
- Once the desired correctness properties of the system have been formally expressed in this way, they can then be verified using a model checker.
- This performs an exhaustive analysis of the system model, for each property either concluding that it is satisfied or, if not, providing a counterexample illustrating why it is violated.
Probabilistic model checking
This is a generalization of model checking for the verification of systems that exhibit stochastic behaviour.

- The models that are constructed and analysed are augmented with quantitative information regarding the likelihood that transitions occur and the times at which they do so.
- In practice, these models are typically Markov chains or Markov decision processes.
- To model systems of reactions at a molecular level, the appropriate model is continuous-time Markov chains (CTMCs), in which transitions between states are assigned (positive, real-valued) rates. These values are interpreted as the rates of negative exponential distributions.

Properties of CTMCs are, like in non-probabilistic model checking, expressed in temporal logic, but are now expressed quantitative in nature.
- For this, one uses probabilistic temporal logics such as continuous stochastic logic (CSL).
- For example: rather than verifying that ‘the protein always eventually degrades’, using CSL allows us to ask ‘what is the probability that the protein eventually degrades?’ or ‘what is the probability that the protein degrades within t hours?’
Plot showing the probability for each possible outcome of the faulty transducer pair, after $T$ seconds.
Visual DSD has been used to model and analyse a wide range of DNA species in the interference between DNA strands. Examples of queries include the probability of reaching a given state within a particular time, or the probability of reaching an undesirable state due to loading into a spreadsheet. Continuous-time Markov chains can be exported as a starting point for construction of the systems. Chemical reaction networks can be exported to SBML (Hucka et al., 2003) and a range of other devices including catalytic gates and populations can also be plotted. The tool can also output nucleotide sequences of Fig. 1) to a graph-based visualization of the entire chemical reaction system, using the graphical notation introduced in Lakin, 2007). These schemes for simulating arbitrary chemical systems (Phillips and 2011) and a range of other devices including catalytic gates and pseudoknot and multiloop junctions are not currently supported; however, some of these structures, including hairpins, are currently under development for a future release. We have found that the tool tool embodies the scientific workflow and can increase efficiency and productivity of DNA device design and analysis by allowing users to simulate systems before attempting to build them in the lab, thereby saving time and lab resources. It also enables model checking (Kwiatkowska, 2007) and the time series data from simulations can be saved in CSV format for verification using stochastic model checking (Kwiatkowska, 2007).

DISCUSSION

2.3 Visualization and export

The example shown implements a simple transducer gate: The Compilation tab on the right-hand side displays output from the compiler, in this case a visualization of all the individual reactions. The Simulation tab shows time-course plots and data tables from stochastic and deterministic simulations. The Analysis tab shows various representations of the continuous-time Markov chain.

Screenshot of the Visual DSD tool:

- Code entry box on the left and Output tabs on the right.
Along the top of the screen are options to select example programs, adjust the semantics and control the simulator.

- The Compilation tab on the right-hand side displays output from the compiler, in this case a visualization of all the individual reactions.
- The Simulation tab shows time-course plots and data tables from stochastic and deterministic simulations.
- The Analysis tab shows various representations of the continuous-time Markov chain.
The Visual DSD tool comes with a number of example systems implemented using DNA molecules. These are accessible from the drop-down menu labeled “Examples” in the top-left corner of the Silverlight user interface.

**Built in Visual DSD Examples:**

The Catalytic example is an implementation of the entropy-driven catalytic gate from (Zhang, Turberfield, Yurke, & Winfree, 2007).

The Lotka example is the Lotka-Volterra predator-prey oscillator.

The Mapk example models a mitogen-activated protein kinase (MAPK) signaling cascade (Huang & Ferrel, 1996)

The Migrations example serves to demonstrate the branch migration rate model (Zhang & Winfree, 2009).
Using the catalytic gate as a running example:

- Selecting this populates the “Code” tab in the left-hand pane with the text of the example program.

- The text of this program begins with a directive to the simulator telling it the duration of the simulation run and how many sample data points to use.

- The next line specifies a “scaling factor” which the system uses to automatically scale up from molar concentrations to populations of individuals, for the stochastic simulation.

- The third and fourth lines declare two domains with specified binding and unbinding rates.

- The final element of the program is a collection of DNA molecules with their respective concentrations.

- Now that we have a program to run, clicking on the “Compile” button performs the compilation into chemical reactions.
The “Input” tab visualises the initial DNA molecules in the system (exactly as they were entered in the code) using a common graphical notation.

Within the “Compilation” tab, the “Species” tab uses the same graphical notation but provides a list of all of the species which could possibly be produced by reactions from the initial species presented in the input program.

The “Reactions” and “Graph” tabs display the set of possible reactions between the various DNA species.
- The “Reactions” tab lists the reactions.
- The “Graph” tab visualizes them as a reaction network.
The outputs of the graphical tabs for the catalytic gate example:

Labeled Nodes:
- Each labelled node in the “Graph” tab denotes a species. (The initial species are represented with a bold outline.)

Unlabeled Nodes:
Each unlabelled node represents a reaction, which may or may not be reversible, with edges connected to reactant and product species:
- For irreversible reactions, edges with no arrows denote reactants, while edges with hollow arrows denote products.
- For reversible reactions, hollow and solid arrows are used to distinguish between the products of the forward and reverse reactions, respectively.
The “Plot” tab produces a real-time graph of the concentrations (or populations) of certain species:

- The species to plot can be specified in the program but our example gives no such directives – in this case the default behaviour is to plot the populations of all species.
- The chart window can be dragged using the mouse and zoomed in and out using the scroll wheel.
- Along the top of the plot window is a collection of buttons which give more control over the plot.
- Clicking on the button for a particular species toggles the visibility of the relevant line in the plot.
- There are also buttons to show all plots and to hide all plots. This selection bar can itself be hidden or moved to dock at the right-hand side of the screen instead of at the top.

When the simulation terminates or is paused:

- The “Initial state” and “Last state” tabs are populated with a visualization of the initial and final states of the simulation run, respectively.
- This includes a graphical visualization of each molecular species, along with their populations.