Homework 3 – ECS 289, Winter 2016

1. **Fast stable computation.** Design a CRC that stably computes $f(n_1, n_2) = n_2$ if $n_1 > 0$ and $f(n_1, n_2) = 0$ otherwise. It should reach a stable configuration in expected time $O(\log n)$, where $n = n_1 + n_2$, and the volume is n.

2. **Chemical caucusing.** Imagine we have two candidates, B and H, being voted on by $n \in \mathbb{Z}^+$ caucus-goers. If B and H supporters encounter each other, one becomes undecided. Undecided voters are swayed by decided voters:

$$\begin{aligned} B + H &\rightarrow B + U \\ B + H &\rightarrow H + U \\ B + U &\rightarrow 2B \\ H + U &\rightarrow 2H \end{aligned}$$

   I lived in Iowa for 12 years, and this is a mostly accurate summary of how the caucus works.

   (a) Show that for any initial configuration $\mathbf{i}$ with $\|\mathbf{i}\| = n$ and $\mathbf{i}(B) + \mathbf{i}(H) > 0$, for all $\mathbf{c}$ such that $\mathbf{i} \Longrightarrow \mathbf{c}$, there is $\mathbf{o}$ such that $\mathbf{c} \Longrightarrow \mathbf{o}$ and either $\mathbf{o}(B) = n$ or $\mathbf{o}(H) = n$. That is, the CRN is guaranteed eventually to reach a consensus.

   (b) **extra credit:** Show that if the initial configuration is $\mathbf{i} = \{0.51n\ B, 0.49n\ H\}$, then for sufficiently large n, with probability at least 99%, the CRN reaches the configuration $\{nB\}$. That is, the initial majority probably wins.

   (c) Deriving how the expected time scales as a function of n is remarkably difficult. (Or, there is a short, elegant proof that has not yet been discovered.)

   Run simulations for different initial configurations and compute the time to stabilize to a consensus.[1] How does the time scale with different initial population sizes, and how is it affected by different initial distributions of B, H, and U?

3. **Reachability versus fair executions.** We observed that the definition of stable computation did not say that a CRC *will* stabilize to the correct answer, merely that it always *could* (using the reachability relation $\Longrightarrow$ to define what could happen). We then proved that a CRC stably computing a function under the reachability definition, when simulated under the kinetic model, actually *will* stabilize to the correct output with probability 1. Now we

[1] Since the CRN has nothing but 2-reactant/2-product/unit-rate-constant reactions, you don't have to worry about simulating the full chemical kinetic model. Just pick a pair of molecules at random to "interact". The interaction counts whether the molecules react or not, e.g., if you pick two copies of B, this counts as an interaction even though there is no reaction $B + B \rightarrow \ldots$. Count how many interactions are needed to stabilize, divide this by n, and call this the "time" (since we expect about n interactions per unit time in a solution with volume n and n molecules).

discuss an alternate nonprobabilistic way to formalize that the CRC *will* stabilize to the correct output. It defines "fair" executions and requires that every fair execution do the correct thing.

A configuration $\mathbf{c}$ is *terminal* if no reaction is applicable to it. A finite execution $\mathcal{E} = (\mathbf{c}_0, \ldots, \mathbf{c}_k)$ is *fair* if $\mathbf{c}_k$ is terminal, and an infinite execution $\mathcal{E} = (\mathbf{c}_0, \mathbf{c}_1, \ldots)$ is *fair* if, for all $\mathbf{c}$, if there are infinitely many $i \in \mathbb{N}$ such that $\mathbf{c}_i \Longrightarrow \mathbf{c}$, then there are infinitely many $j \in \mathbb{N}$ such that $\mathbf{c}_j = \mathbf{c}$. In other words, every configuration that is infinitely often reach*able* in $\mathcal{E}$ is infinitely often reach*ed*.[2]

Show that a CRC $\mathcal{C} = (\Lambda, R, \Sigma, Y, \mathbf{s})$ stably computes a function f under the definition given in lecture if and only if, for every valid initial configuration $\mathbf{i} \in \mathbb{N}^\Lambda$, every fair execution $\mathcal{E} = (\mathbf{i}, \ldots)$ contains an output stable configuration $\mathbf{o} \in \mathbb{N}^\Lambda$ such that $\mathbf{o}(Y) = f(\mathbf{i} \upharpoonright \Sigma)$.

4. **Impossibility of stable multiplication (extra credit).** I claimed in class that no non-semilinear set or function can be stably decided/computed by a CRN. In this problem you will use one of these facts to prove the other.

   You may assume that no CRD stably decides the set $\left\{ (n_1, n_2) \in \mathbb{N}^2 \;\middle|\; n_1 = (n_2)^2 \right\}$. Use this fact to show that no CRC stably computes the function $f(n_1, n_2) = n_1 \cdot n_2$.

[2]This concept is borrowed from distributed computing. A distributed algorithm runs over a network in which the order in which various nodes will receive messages from each other is unknown. It would be too restrictive to imagine that the algorithm should work under *any* schedule of message delivery. For example, there there are three nodes a, b, c, if the scheduler always delivers messages to a and b before it delivers them to c, and if every message received by a triggers an immediate response message to b and vice versa, then c will be "starved" under this unfair schedule.

There are many definitions of fair schedules in CRNs. This is the one most commonly used. Others look different but are equivalent; for instance, requiring that if there are infinitely many $i \in \mathbb{N}$ such that $\mathbf{c}_i \Longrightarrow^1 \mathbf{c}$ (i.e., $\mathbf{c}_i$ can reach to $\mathbf{c}$ by a single reaction), then there are infinitely many $j \in \mathbb{N}$ such that $\mathbf{c}_j = \mathbf{c}$. Others that look reasonable actually define a weaker notion; for instance, requiring that every reaction that is infinitely often applicable is infinitely often applied. By "weaker", we mean that more executions are fair under the latter definition, and some of the new executions fail to get the answer correct.