CORRECTION March 13th 2013

MPRI C2-19 Examination, Part II, F. Fages
Computational Methods for Systems and Synthetic Biology
Compiling Imperative Programs into Biochemical Reactions

We consider the possibility of compiling imperative programs on a fixed number of variables, into systems of biochemical reactions using mass action kinetics and only two kinetic rate constants s and f, for fast and slow reactions respectively. By convention, reactions without indication of their kinetics will be slow.

For these transformations, we consider an intermediate language of *conditional reactions* with preconditions. These conditions are logical expressions over Boolean variables associated to each molecular species (noted with the same letter by abuse of notation) and obtained by the standard Boolean abstraction of molecular concentration for some given threshold value θ:

$$A = \begin{cases} 1 \text{ if } [A] \geq \theta \\ 0 \text{ if } [A] < \theta \end{cases}$$

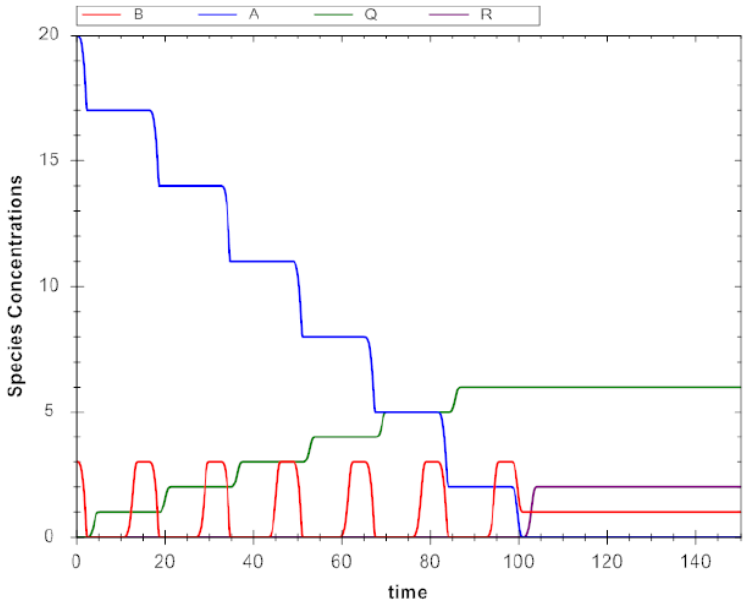
For instance, the following program for the Euclidean division of A by B, will be first compiled in a conditional reaction program where initially Q is zero and C is initially of a unit amount:

```
Q:=0
while A>=B do
begin
   A:=A-B;
   Q:=Q+1;
end;
R:=A
```

```
A + B => D
C => Q + E precondition (not B)
D => F precondition (not C)
E => G precondition (not D)
F => B precondition (not E)
G => C precondition (not F)
D => R precondition (B and not A)
```

and then into a system of biochemical reactions with mass action kinetics.

The execution with $[A] = 20$ and $[B] = 3$ produces the result $[R] = 6$ as follows:



In the sequel, we consider reactions with an arbitrary number of reactants.

1 Compiling Preconditions

We have to compile the preconditions of the intermediate language into biochemical reactions. For instance, for a simple precondition testing the presence of a molecule D

```
A + B => C precondition D
```

we can just add it as a catalyst to the reaction

```
A + B + D => C + D
```

This transformation does not alter the concentration of D and allows the reaction to proceed only when D is present.

1.1 Conjunction

Give a compilation scheme for

```
A + B => C precondition (D and E)
```

Answer:

```
A + B + D + E => C + D + E
```

1.2 Disjunction

Give a compilation scheme for

```
A + B => C precondition (D or E)
```

Answer:

```
A + B + D => C + D
A + B + E => C + E
```

1.3 Constructive negation

A negation in a precondition amounts to test the absence of a molecular species which cannot be directly done in a biochemical reaction. The idea is to introduce a witness molecule A' for the absence of A without affecting A, using the following slow and fast mass action law kinetic reactions:

```
MA(s) for _ => A'
MA(f) for A + A' => A
MA(f) for 2*A' => A'
```

Write the ordinary differential equation associated to this reaction system and express the relationship between $[A]$, $[A']$, s and f at equilibrium when $[A]$ is, respectively, zero and high (i.e. greater than $[A']$).

Answer:

$d[A]/dt = 0$

$d[A']/dt = s - f * [A] * [A'] - f * [A']^2$

Note that [A] is constant.

At equilibrium, we have $s = f * [A']^2 + f * [A] * [A']$ *so* $[A']$ *cannot be exactly zero*

$$[A'] = \frac{-f * [A] + \sqrt{f^2 * [A]^2 + 4 * f * s}}{2 * f}$$

When $[A] = 0$ *we have* $[A'] = \sqrt{s/f}$.

When $[A] > [A']$ *we have* $s > 2 * f * [A']^2$ *hence* $[A'] < \sqrt{s/(2 * f)}$.

2 Compiling Arithmetic

Now, let us consider assignment instructions. For the copy instruction, `B:=A`, compiling it with just one reaction `A => B` would destroy A. On the other hand, the reaction `A => A+B` would increase B at each increment of A. In order to localize the computation for the copy, we use the following conditional reactions

```
A => C   precondition (G)
G => _   precondition (not A)
C => A+B precondition (not G)
```

where G is a start signal molecule for executing the instruction and which is consumed in the process.

2.1 Multiplication by Two

Modify the previous conditional reactions in order to implement `B:=2*A`.

Answer:
We just have to replace the last conditional reaction by

```
C => A+2*B precondition (not G)
```

2.2 Division by Two

Modify the previous conditional reactions to implement the integer division `B:=A/2`.

Answer:
We replace the last conditional reaction by

```
2*C => 2*A+B precondition (not G)
C => A       precondition (not 2C)
```

where the precondition (not 2C) is defined in the obvious manner as above.

2.3 Comparison

Give conditional reactions to implement the comparison `R:=(A>B)` with start signal molecule G.

Answer:
Let us assume that A and B are first copied with the instructions `Ac:=A` and `Bc:=B`. The comparison between Ac and Bc can be implemented by the following condition reaction program:

```
Ac+Bc => C   precondition (G)
G => R       precondition (Ac and not Bc)
```

Same question for `R:=(A>=B)`.

Answer:
The last reaction is now

```
G => R       precondition (not Bc)
```

3 Compiling Control Flows

3.1 Sequential composition

A reaction ends when one of its reactants is exhausted. Give the preconditions to enforce the sequential execution of the following chain of reactions:

```
A => B then B => C then C => D
```

Answer:

```
A => B
B => C precondition (not A)
C => D precondition (not B)
```

3.2 Branching statement

Give conditional reactions to compile

```
if (A) then P else if (B) then Q else R
```

with start signal molecule G and where P, Q and R are programs with start signal molecule Gp, Gq, Gr respectively.

Answer:

```
G => Gp precondition (A)
G => Gq precondition (B and not A)
G => Gr precondition (not A and not B)
```

3.3 Looping statement

Give conditional reactions to compile

```
while (A>B) do A:=A-B
```

with start molecule G
Answer:

```
A+B => C   precondition (G)
C => D     precondition (A and not B)
D => B     precondition (not C)

C => E     precondition (not A)
E => A+B
```