Towards Programmable Chemistries[⋆]

Dandolo Flumini[1], Mathias S. Weyland[1], Johannes J. Schneider[1], Harold Fellermann[2], and Rudolf M. Füchslin[1]

[1]School of Engineering, Zurich University of Applied Sciences, Technikumstrasse 9, 8400 Winterthur, Switzerland, {flum,weyl,scnj,furu}@zhaw.ch
[2] School of Computing, Newcastle University, Newcastle-upon-Tyne, NE4 5TG, harold.fellermann@ncl.ac.uk

Abstract. We provide a practical construction to map (slightly modified) *GOTO*-programs to chemical reaction systems. While the embedding reveals that a certain small fragment of the chemtainer calculus is already Turing complete, the main goal of our ongoing research is to exploit the fact that we can translate arbitrary control-flow into real chemical systems. We outline the basis of how to automatically derive a physical setup from a procedural description of chemical reaction cascades. We are currently extending our system in order to include basic chemical reactions that shall be guided by the control-flow in the future.

Keywords: Programmable Chemistry · Compartmentalization · Biochemical Engineering · Theoretical Computer Science

1 Introduction

In order to "program" chemical reaction systems, we provide a construction to map procedural control-flow to chemical reaction systems.

The computational framework that we use to represent arbitrary control flow is (slightly modified) GOTO programs. In section 3 we give a short introduction to the GOTO formalism. The results presented in this section are standard. The related result presented in Lemma 1 (section 4) is also considered to be known; however, no corresponding reference was found.

The system to represent chemical reaction systems is the chemtainer calculus. In section 2, we give a short introduction to the relevant notions of the formalism. For a more detailed account, the reader is referred to [16].

Section 4 is the main contribution of the present work. We discuss the actual embedding of arbitrary GOTO programs into the chemtainer calculus. We also discuss a variation of the construction that solves some issues that render the original embedding unsuitable for practical use in a "programmable chemistry" setting. All of the work presented is original research.

In section 5 we discuss our results, ongoing research, and future directions.

[⋆] Funded by Horizon 2020 Framework Programme of the European Union (project 824060)

1.1 Related Work

Chemical reaction systems are formally described by chemical reaction networks (CRNs) [1, 2]. They can be used to facilitate the analysis of artificial chemistries [3] and real chemistries. To name a few applications, CRNs have been used to predict reaction paths [4], to model spontaneous emergence of self-replication [5], to synthesize optimal CRNs from prescribed dynamics [6], and to design asynchronous logic circuits [7].

In the European Commission-funded project ACDC, we are developing a programmable artificial cell with distributed cores. An important feature of the systems studied in the context of ACDC is compartmentalization. CRNs alone are not suitable to model compartmentalization. However, formalisms with the ability to express compartmentalization have been developped [8–16]. Our systems can be described particularly well with the chemtainer calculus [16], one of the aforementioned formalisms. Thus, the chemtainer calculus is chosen for emulating computations with chemical reaction systems in the present work.

2 The Chemtainer Calculus

As discussed in the previous section, the chemtainer calculus is a formal calculus capable of describing compartmentalized reaction systems [16]. In this section, the subset of chemtainer calculus necessary for the emulation of computations with chemical reaction systems is introduced. It consists of the following objects:

- **Molecules:** Objects that can undergo reactions as specified in a CRN. Capital letters $(A, B, C, ...a)$ are used to denote molecules.
- **Chemtainers:** Compartments that contain objects (including other chemtainers). The symbols $\Lbag$ and $\Rbag$ are used to indicate objects enclosed in chemtainers.
- **Address tags:** Tags that are in solution or attached to a chemtainer. Lower case greek letters $(\tau, \sigma, ...)$ are used to denote tags.

The notion of space is implemented with discrete locations $(x, y, m_i, ...)$ at which objects reside. A number of instructions are used to alter the system state. They are introduced below by examples:

- **feed**$(x,\ A,\ 3)$: A chemtainer containing 3 instances of molecule A is fed into location x. Starting from an empty state, this yields:

$$\emptyset \rightarrow x : \Lbag\, 3A\, \Rbag$$

- **feed_tag**$(x,\ \sigma,\ 1)$: One tag with address σ is fed into location x. Again presuming an empty initial state, the instruction yields:

$$\emptyset \rightarrow x : 1\sigma$$

– **tag**(x): Decorate chemtainer in location x with the tags surrounding the chemtainer:

$$x : 1\sigma + \langle\!\langle\, 3A \,\rangle\!\rangle \to x : 1\sigma\langle\!\langle\, 3A \,\rangle\!\rangle$$

– **move**(σ, x, y): Move any tag σ (including potentially attached chemtainers) from location x to location y:

$$x : 1\sigma\langle\!\langle\, 3A \,\rangle\!\rangle \to y : 1\sigma\langle\!\langle\, 3A \,\rangle\!\rangle$$

– **fuse**(x): Fuse chemtainers in location x:

$$x : \langle\!\langle\, 2A \,\rangle\!\rangle + \langle\!\langle\, 2B \,\rangle\!\rangle \to x : \langle\!\langle\, 2A + 2B \,\rangle\!\rangle$$

– **flush**(x): Remove any objects from location x:

$$x : \langle\!\langle\, 2A + 2B \,\rangle\!\rangle \to x : \emptyset$$

– **burst**(x): Burst chemtainers in location x, releasing any contained molecules and leaving behind empty chemtainers:

$$x : \langle\!\langle\, 2A + 2B \,\rangle\!\rangle \to 2A + 2B + \langle\!\langle\quad \rangle\!\rangle$$

Chemtainer programs are a sequence of such instructions that alter the system state.

3 GOTO-Programs

We work with a slight variation of the standard syntax of *GOTO* programs as presented in [17]. Our syntactic building blocks are as follows:

– Countably many **variables** $x_0, x_1, x_2, \ldots,$
– **literals** $0, 1, 2, \ldots$ for nonegative integers,
– **markers** $M_1, M_2, M_3, \ldots,$
– **separator** symbols $=, :,$
– **operator** and **relation**symbols $+, -, >,$
– and **keywords** GOTO, IF, THEN, HALT.

Instructions of GOTO-programs take one of the following forms:

– **Assignments:** $x_i := x_i \pm c$ where $i \in \mathbb{N}$, c is a literal (for a nonnegative integer), and $\pm$ stands for either $+$ or $-$.
– **Jumps:** GOTO M_k where $k \in \mathbb{N}$
– **Conditional Jumps:** IF $x_i > 0$ THEN GOTO M_k where $i, k \in \mathbb{N}$
– **Halt instruction:** HALT

A GOTO-program is a finite sequence of instructions, each of which is given with a unique label of the form M_i where $i \in \mathbb{N}$. In order to enhance readability, we will generally write up GOTO-programs in vertical order

$$\begin{gathered} M_1 : I_1 \\ \vdots \\ M_k : I_k. \end{gathered}$$

In section 4, we observe how every computation of a *GOTO*-program can be "emulated" by the state-transitions of a suitably constructed chemical reaction system. As a result, we obtain that a suitable chemical reaction system can emulate every computation (in the sense of Turing completeness).

To match *GOTO*-program computations with state-transitions of a chemical reaction system, we introduce an operational semantics for the *GOTO* language that captures the idea of a global state being mutated while instructions are executed sequentially.

The state of a *GOTO*-program-computation is completely determined by a marker (stating the "current" instruction) and the values held by relevant variables (i.e. all variables occurring in the program at hand). Thus, for a given *GOTO*-program P with variables $x_0, \dots, x_n$ and markers $M_1, \dots, M_k$, the state of a computation can be modelled as a tuple $(X, \boldsymbol{y})$ where $X \in \{M_1, \dots, M_k, \perp\}$ indicates the "current" instruction and $\boldsymbol{y} \in \mathbb{N}^{n+1}$ holds the values stored in the variables $x_0, \dots, x_n$. Cases where $X = \perp$ indicate that the computation has halted. The (deterministic) operational semantics is given by the following transition relation:

Let P be any *GOTO*-program with variables among $x_0, \dots, x_n$, let $y_0, y_1, \dots$ range over natural numbers, and let l_c stand for the literal associated with a natural number c.

- If $M_i : x_r := x_r \pm l_c$ is part of P, and if the following line is labelled with marker M_k, then

$$(M_i, y_0, \dots, y_r, \dots, y_n) \xrightarrow{P} \begin{cases} (M_k, y_1, \dots, 0, \dots, y_n) & \text{if } y_r \pm c \leq 0 \\ (M_k, y_1, \dots, y_r \pm c, \dots, y_n) & \text{otherwise.} \end{cases}$$

- If $M_i : x_r := x_r \pm l_c$ is the last line of P, then

$$(M_i, y_0, \dots, y_r, \dots, y_n) \xrightarrow{P} \begin{cases} (\perp, y_1, \dots, 0, \dots, y_n) & \text{if } y_r \pm c \leq 0 \\ (\perp, y_1, \dots, y_r \pm c, \dots, y_n) & \text{otherwise.} \end{cases}$$

- If $M_i : \ \mathsf{GOTO}\ M_k$ is a line of P, then

$$(M_i, y_0, \dots, y_n) \xrightarrow{P} \begin{cases} (M_k, y_0, \dots, y_n) & \text{if } M_k \text{ is a label in } P \\ (\perp, y_0, \dots, y_n) & \text{otherwise} \end{cases}$$

– If M_i : $\mathsf{IF}\ x_j > 0\ \mathsf{THEN}\ \ \mathsf{GOTO}\ M_k$ is a line in P, and if $y_j > 0$, then

$$(M_i, y_0, \dots, y_n) \xrightarrow{P} \begin{cases} (M_k, y_0, \dots, y_n) & \text{if } M_k \text{ is a label in } P \\ (\bot, y_0, \dots, y_n) & \text{otherwise.} \end{cases}$$

– If M_i : $\mathsf{IF}\ x_j > 0\ \mathsf{THEN}\ \ \mathsf{GOTO}\ M_k$ is a line in P, and if $y_j = 0$ and the following line is labelled with marker M_k, then

$$(M_i, y_0, \dots, y_n) \xrightarrow{P} (M_k, y_0, \dots, y_n)$$

– If M_i : $\mathsf{IF}\ x_j > 0\ \mathsf{THEN}\ \ \mathsf{GOTO}\ M_k$ is the last line in P, and if $y_j = 0$, then

$$(M_i, y_0, \dots, y_n) \xrightarrow{P} (\bot, y_0, \dots, y_n).$$

– If M_k : HALT is a line in P, then

$$(M_k, y_0, \dots, y_n) \xrightarrow{P} (\bot, y_0, \dots, y_n)$$

– No other cases are considered.

Further, we write $x \xrightarrow{P,1} y$ if $x \xrightarrow{P} y$, and $x \xrightarrow{P,n+1} y$ if there is a state z such that $x \xrightarrow{P,n} z$ and $z \xrightarrow{P} y$. Since labels in G-programs are unique, the resulting transition system is deterministic in the sense that $x \xrightarrow{P} y \wedge x \xrightarrow{P} y'$ implies $y = y'$ for all states x, y and y'. Therefore it is meaningful to write $x^{P,n}$ for the unique state of P that satisfies $x \xrightarrow{P,n} x^{P,n}$.

Based on the given transition relation we can introduce the usual denotational semantics for GOTO-programs; for every GOTO-program P and every $k \in \mathbb{N}$ the (partial) function $[P,k] : \mathbb{N}^k \to \mathbb{N}$ is given from:

$$[P,k](y_1, \dots, y_k) = y \Leftrightarrow \exists n, \boldsymbol{z}((m, (0, y_1, \dots, y_k, \mathbf{0}))^{P,n} = (\bot, (y, \boldsymbol{z})) \qquad (1)$$

where m denotes the marker of the first line in P and $\mathbf{0}$ represents a sequence of zeros, so that all variables in P are initialized properly. The equivalence stated in (1) means that we evaluate a *GOTO* program P as a k-ary $[P,k]$ function as follows:

– Initialize the variables $x_1, \dots, x_k$ with the input values (additional variables of P are initialized with 0).
– Execute the program P starting with the first instruction and according to the state transitions given above.
– If the execution halts, read the variable x_0 to obtain the output of the function $[P,k]$ for the given input vector.

It is well known that *GOTO* is Turing complete with respect to this semantics [17].

4 Emulating Computations with Chemical Reaction Systems

In this section, we rely on the notions of artificial cellular matrices, the chemtainer calculus as introduced in section 2, and chemtainer programs. We refer the reader to [16] for further details.

We demonstrate how given any *GOTO*-program P, we can construct an artificial cellular matrix M together with a chemtainer program to simulate P. In a first step, we will show how to match any state x of a *GOTO*-program to a global state $\ulcorner x \urcorner$ of the chemtainer calculus, and then we will describe how to translate the *GOTO* program P into a corresponding chemtainer program $\langle\!\langle P \rangle\!\rangle$, so that all state transitions of P are simulated in M (Proposition 1).

4.1 Matching States of *GOTO*-Program-Computations with Global States of the Chemtainer Calculus

For a given *GOTO*-program P with variables $x_0, \ldots, x_n$ and markers $M_1, \ldots, M_k$, we identify states $(M_i, \boldsymbol{y})$ of the computations of P with global states $\ulcorner(M_i, \boldsymbol{y})\urcorner$ of the chemtainer calculus as follows: We use tags $\tau_0, \ldots, \tau_n$, a special "control-flow" tag σ, and locations $\tilde{m_0}, m_1, \tilde{m_1} \ldots, m_k, \tilde{m_k}$ as well as a special location *halt* to stipulate

$$\ulcorner(M_i, \boldsymbol{y})\urcorner = \\ \tilde{m_0} : 0 \circ \tilde{m_1} : 0 \circ m_1 : 0 \circ \cdots \circ m_i : \tau_0^{y_0} \ldots \tau_n^{y_n} (\!(\, 0 \,)\!) \circ \cdots \circ m_k : 0 \circ \mathit{halt} : 0$$

and

$$\ulcorner(\perp, \boldsymbol{y})\urcorner = \tilde{m_0} : 0 \circ \tilde{m_1} : 0 \circ m_1 : 0 \circ \cdots \circ m_k : 0 \circ \mathit{halt} : \tau_0^{y_0} \ldots \tau_n^{y_n} (\!(\, 0 \,)\!)$$

where $\tau_i^{y_i}$ stands for the y_i fold repetition of τ_i. An illustration of the correspondence is shown in figure 1.

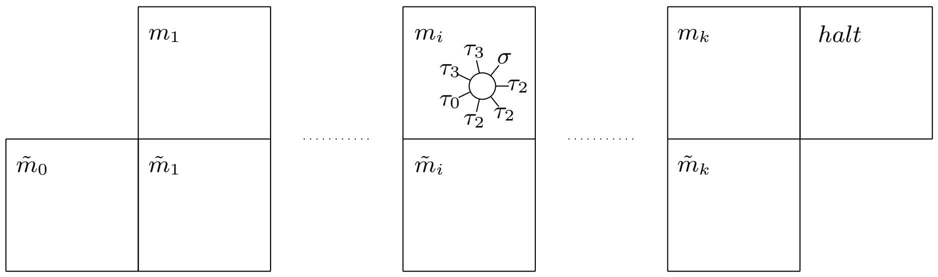

Fig. 1. An illustration of the state $(M_i, 1, 3, 2, 0, \ldots, 0)$ of a *GOTO*-program-computation interpreted as a global state of the chemtainer calculus.

4.2 The Construction of the Chemtainer Program

The mapping of states defined in section 4.1 now enables us to specify a construction enabling us to translate any *GOTO* program P to a chemtainer program $\langle\langle P\rangle\rangle$ that emulates the computation of P. Our general strategy is first to associate lines (i.e., tagged instructions) $M_j : I_j$ of the *GOTO*-language to simple chemtainer programs $\langle M_j : I_j\rangle$, and then to show how the mapping can be extended to translate complete *GOTO* programs consisting of several lines of code.

The basic chemtainer programs $\langle M_j : I_j\rangle$ are specified by case analysis as follows:

$$\begin{aligned}
\langle M_j : x_i := x_i + l_c\rangle &= \mathbf{feed\_tag}(m_j, \tau_i, c); \mathbf{tag}(m_j)\\
\langle M_j : x_i := x_i - l_c\rangle &= \mathbf{feed\_tag}(m_j, \bar{\tau}_i, c)\\
\langle M_j : \mathsf{GOTO}\ M_i\rangle &= \mathbf{move}(\sigma, m_j, \tilde{m}_{i-1})\\
\langle M_j : \mathsf{IF}\ x_r > 0\ \mathsf{THEN\ GOTO}\ M_i\rangle &= \mathbf{move}(\tau_r, m_j, \tilde{m}_{i-1})\\
\langle M_i : \mathsf{HALT}\rangle &= \mathbf{move}(\sigma, i, halt).
\end{aligned}$$

Next, we translate *GOTO* programs that are composed of several instructions. In favor of a more concise description, we will here and henceforth assume (without loss of generality) that *GOTO* program-lines are marked in order $M1, M2, M3, \ldots,$ and that jump instructions may only lead to markers that are present in the program at hand. We thus assume, that the given program P is of the form

$$\begin{gathered}
M_1 : I_1\\
\vdots\\
M_k : I_k,
\end{gathered}$$

and we stipulate $\langle\langle P\rangle\rangle$ for the following chemtainer program:

$$\begin{aligned}
&\langle M_1 : I_1\rangle;\\
&\langle M_2 : I_2\rangle;\\
&\ \ \vdots\\
&\langle M_k : I_k\rangle;\\
&\ \mathbf{move}(\sigma, m_1, \tilde{m}_1); \ldots; \mathbf{move}(\sigma, m_k, \tilde{m}_k);\\
&\ \mathbf{flush}(m_1); \ldots; \mathbf{flush}(m_k)\\
&\ \mathbf{move}(\sigma, \tilde{m}_0, m_1); \mathbf{move}(\sigma, \tilde{m}_1, m_2); \ldots; \mathbf{move}(\sigma, \tilde{m}_k, halt);
\end{aligned}$$

Now, given a global state S of the chemtainer calculus, we write $S^{P,n}$ for the global state (in order to obtain determinism, we here need to restrict the original rule number 56 of the chemtainer calculus (as introduced in [16]) to only be admissible if the respective location is empty.) that results from S when the program $\langle\!\langle P \rangle\!\rangle$ is applied exactly n times. In the next lemma, we state that the correspondence declared in section 4.1 is a simulation relation.

Proposition 1. *Let P be any GOTO-program of the form*

$$\begin{gathered} M_1 : I_1 \\ \vdots \\ M_k : HALT \end{gathered}$$

such that all jump instructions in P refer to a marker $M_1, \ldots, M_k$. If x is a state in a computation of P, then for all $n \in \mathbb{N}$,

$$\ulcorner x^{P,n} \urcorner = \ulcorner x \urcorner^{\langle\!\langle P \rangle\!\rangle, n}.$$

Proof. Let P and x be as stated in the claim. Applying induction on n, we only need to prove that $\ulcorner x^{P,1} \urcorner = \ulcorner x \urcorner^{P,1}$. Let x be

$$(M_i, \boldsymbol{y})$$

where $\boldsymbol{y} = y_0, \ldots, y_n$. The proof proceeds by case distinction on the instruction I_i. We can assume that I_i is not the last instruction in P if I_i is not the $HALT$ instruction.

– If I_i is $x_j := x_j + l_c$, then $x^{P,1}$ is $(M_{i+1}, y_0, \ldots, y_j + c, \ldots, y_n)$ and thus

$$\begin{aligned} \ulcorner x^{P,1} \urcorner = \tilde{m}_0 : 0 \circ m_1 : 0 \circ \tilde{m}_1 : 0 \circ \ldots \circ m_{i+1} : \tau_0^{y_0} \ldots \tau_j^{y_j + c} \\ \ldots \tau_n^{y_n} \llparenthesis\, 0 \,\rrparenthesis \circ \cdots \circ m_k : 0 \circ halt : 0. \end{aligned}$$

When running $\langle\!\langle P \rangle\!\rangle$ with initial state

$$\begin{aligned} \ulcorner x \urcorner = \tilde{m}_0 : 0 \circ m_1 : 0 \circ \tilde{m}_1 : 0 \circ \ldots \\ \circ m_i : \tau_0^{y_0} \ldots \tau_n^{y_n} \llparenthesis\, 0 \,\rrparenthesis \circ \cdots \circ m_k : 0 \circ halt : 0 \end{aligned}$$

the right number of tags are attached to the chemtainer in the "first part" of the program, and the chemtainer is relocated to m_{i+1} in two steps resulting in the same global state

$$\begin{aligned} \ulcorner x \urcorner^{\langle\!\langle P \rangle\!\rangle, 1} = \tilde{m}_0 : 0 \circ m_1 : 0 \circ \tilde{m}_1 : 0 \circ \cdots \circ m_{i+1} : \tau_0^{y_0} \ldots \tau_j^{y_j + c} \\ \ldots \tau_n^{y_n} \llparenthesis\, 0 \,\rrparenthesis \circ \cdots \circ m_k : 0 \circ halt : 0. \end{aligned}$$

– The case where I_i is $x_j := x_j - l_c$ works essentially like the previous case, with the difference that no tagging instruction is introduced and the released tags bind to the complementary tags (that are already attached to the chemtainer).

- If I_i is $\mathsf{IF}\ x_j > 0\ \mathsf{THEN\ \ GOTO}\ M_r$ and $y_j = 0$, then the state $x^{P,1}$ is $(M_{i+1}, \boldsymbol{y})$ and thus $\ulcorner x^{P,1} \urcorner$ is

  $$\tilde{m}_0 : 0 \circ m_1 : 0 \circ \tilde{m}_1 : 0 \circ \cdots \circ m_{i+1} : \tau_0^{y_0} \ldots \tau_n^{y_n} \llparenthesis\, 0 \,\rrparenthesis \circ \cdots \circ m_k : 0 \circ \mathit{halt} : 0$$

  On the other hand, if we run the chemtainer program $\langle\!\langle P \rangle\!\rangle$ with starting state

  $$\ulcorner x \urcorner = \tilde{m}_0 : 0 \circ m_1 : 0 \circ \tilde{m}_1 : 0 \circ \cdots \circ m_i : \tau_0^{y_0} \ldots \tau_n^{y_n} \llparenthesis\, 0 \,\rrparenthesis \circ \cdots \circ m_k : 0 \circ \mathit{halt} : 0,$$

  we note that since $y_j = 0$ there is no τ_j on the surface of the chemtainer, thus no transition in the first "half" of $\langle\!\langle P \rangle\!\rangle$ is effective at all. Thus, the only instructions that have an impact on the global state $\ulcorner x \urcorner$ are $\mathbf{move}(\sigma, m_i, \tilde{m}_i)$ and $\mathbf{move}(\sigma, \tilde{m}_i, m_{i+1})$, resulting in the global state $\ulcorner x \urcorner^{P,1} = \ulcorner x^{P,1} \urcorner$.
- If I_i is $\mathsf{IF}\ x_j > 0\ \mathsf{THEN\ \ GOTO}\ M_r$ and $y_j > 0$, then the state $x^{P,1}$ becomes $(M_r, \boldsymbol{y})$ (we assume that the marker M_r exists in P.), and thus

  $$\ulcorner x^{P,1} \urcorner = \\ \tilde{m}_0 : 0 \circ m_1 : 0 \circ \tilde{m}_1 : 0 \circ \cdots \circ m_r : \tau_0^{y_0} \ldots \tau_n^{y_n} \llparenthesis\, 0 \,\rrparenthesis \circ \cdots \circ m_k : 0 \circ \mathit{halt} : 0.$$

  Accordingly, if we run the chemtainer-program with initial state $\ulcorner x \urcorner$, the instructions of $\langle\!\langle P \rangle\!\rangle$ that actually alter the global state are $\langle M_i : I_i \rangle$ i.e. $\mathbf{move}(\tau_j, m_i, \tilde{m}_{r-1})$ and $\mathbf{move}(\sigma, \tilde{m}_{r-1}, m_r)$, thus we obtain

  $$\ulcorner x \urcorner^{\langle\!\langle P \rangle\!\rangle,1} = \\ \tilde{m}_0 : 0 \circ m_1 : 0 \circ \tilde{m}_1 : 0 \circ \cdots \circ m_r : \tau_0^{y_0} \ldots \tau_n^{y_n} \llparenthesis\, 0 \,\rrparenthesis \circ \cdots \circ m_k : 0 \circ \mathit{halt} : 0$$

  as desired.
- Nonconditional jump instructions are handled exactly like conditional jump instructions where the condition is satisfied.
- If I_i is $HALT$, then $x^{P,1}$ is $(\perp, \boldsymbol{y})$ and thus $\ulcorner x^{P,1} \urcorner$ is

  $$\tilde{m}_0 : 0 \circ \tilde{m}_1 : 0 \circ m_1 : 0 \circ \cdots \circ m_k : 0 \circ \mathit{halt} : \tau_0^{y_0} \ldots \tau_n^{y_n} \llparenthesis\, 0 \,\rrparenthesis$$

  Since the only relevant transition in $\langle\!\langle P \rangle\!\rangle$ when applied to

  $$\ulcorner x \urcorner = \tilde{m}_0 : 0 \circ m_1 : 0 \circ \tilde{m}_1 : 0 \circ \ldots \\ \circ m_i : \tau_0^{y_0} \ldots \tau_n^{y_n} \llparenthesis\, 0 \,\rrparenthesis \circ \cdots \circ m_k : 0 \circ \mathit{halt} : 0$$

  is $\langle M_i : Halt \rangle = \mathbf{move}(\sigma, \mathbf{i}, \mathbf{halt})$ the states $\ulcorner x^{P,1} \urcorner$ and $\ulcorner x \urcorner^{\langle\!\langle P \rangle\!\rangle,1}$ coincide.

□

As a corollary we obtain that any (Turing) computable function can be evaluated by a suitable artificial cellular matrix together with an expression of the chemtainer calculus.

Corollary 1. *Given any recursive funtion $f : \mathbb{N}^k \to \mathbb{N}$, then an artificial cellular matrix, a natural number n and a chemtainer program P exist such that*

$$\begin{aligned}(\tilde{m}_0 : \tau_1^{y_1} \dots \tau_n^{y_n} \llparenthesis\, 0 \,\rrparenthesis \circ m_1 : 0 \circ \dots \circ m_k : 0 \circ \mathit{halt} : 0)^{P,n} \\ = \dots \circ \mathit{halt} : \tau_0^{f(y_1,\dots,y_n)} \dots \llparenthesis\, 0 \,\rrparenthesis\end{aligned}$$

holds whenever $f(y_1, \dots, y_n)$ is defined.

Proof. This follows from Proposition 1 and equation 1. □

4.3 Practical Considerations

While theoretically sound, we identified two main issues of our construction that make it unsuitable for practical use in a "programmable chemistry" setting, both of which have to do with how we encode natural numbers in quantities of molecules:

– If a large number of variables occur in a program, then there might not be a distinct (suitable) molecule for each variable to encode. We call this the "finiteness of molecules problem".
– It is generally infeasible to exactly count numbers of molecules (which means that we cannot effectively read or write variables). We call this the "counting problem".

We can solve the finiteness problem by pointing out that there is a definite natural number N_0 such that every *GOTO*-computation can be realized by a *GOTO*-program with no more than N_0 many variables. This is equivalent to the statement of the next lemma.

Lemma 1. *A natural number N_0 exists, such that for every GOTO program P there is a GOTO program P' with no more than N_0 many variables and $[P, 1] = [P', 1]$.*

Proof. Since the *GOTO* language is Turing complete, a *GOTO* program I exists, such that for a suitable encoding # of *GOTO* programs the equation

$$\lambda x.[I, 2](\#A, x) = [A, 1]$$

holds for every *GOTO* program A. Thus, given any *GOTO* program P a suitable *GOTO* program P' that satisfies the claim is given from

$$\begin{aligned}&M_a : x_2 := x_1 + l_0;\\ &M_b : x_1 := l_{\#P} + l_0;\\ &\quad I\end{aligned}$$

where the markers M_a and M_b do not occur in I. □

In order to solve the counting problem, we have to modify our construction slightly. Since exactly counting the numbers of molecules is not feasible, it is not suitable to represent integer values by exactly matching numbers of specific tags on the surface of a chemtainer. It is, however, simple to measure concentrations and thus to decide whether the concentration of a molecule is "high" or "low" respectively. If not integer values, this enables us to code boolean values effectively. The main idea is as follows:

G_{bool}-programs are obtained by restricting constant and variable values in *GOTO*-programs to 0 or 1 respectively. In contrast to our first embedding, if a variable x_i holds the value 1, this is not translated in the sense that there is exactly one tag τ_i on the surface of some vesicle, but rather that there are many i.e., that the vesicles surface is "covered" with corresponding tags. Accordingly, states of G_{bool}-program computations are identified with global states of the chemtainer calculus similar as in section 4 but the $\tau_k^{y_k}$'s stand for a very short string of τ_k if $y_k = 0$ and a very long string of τ_k otherwise. Similar to the situation shown in figure 1, the state $(M_i, 0, 1, 0, 1, 1, 0, 0, 0, 0)$ is represented by a chemtainer in location m_i with its surface populated by many σ, τ_1, τ_3 and τ_4 tags and none or very few further tags.

The construction of simple chemtainer programs to emulate labeled instructions of G_{bool}-programs then essentially works as with *GOTO*-programs and is given from

$$\langle M_j : x_i := x_i + c \rangle = \begin{cases} \mathbf{feed\_tag}(m_j, \tau_i, \infty); \mathbf{tag}(m_i) & \text{if } c = 1 \\ \epsilon & \text{otherwise} \end{cases}$$

$$\langle M_j : x_i := x_i - c \rangle = \begin{cases} \mathbf{feed\_tag}(m_j, \bar{\tau}_i, \infty); \mathbf{tag} & \text{if } c = 1 \\ \epsilon & \text{otherwise} \end{cases}$$

$$\langle M_j : \mathsf{GOTO}\ M_i \rangle = \mathbf{move}(\sigma, m_j, \tilde{m}_{i-1})$$

$$\langle M_j : \mathsf{IF}\ x_r > 0\ \mathsf{THEN\ GOTO}\ M_i \rangle = \mathbf{move}(\tau_r, m_j, \tilde{m}_{i-1})$$

$$\langle M_i : \mathsf{HALT} \rangle = \mathbf{move}(\sigma, i, halt)$$

where $\mathbf{feed\_tag}(m_j, \tau_i, \infty)$ means that the location m_j is flooded with a non-specific but abundant number of τ_i tags. The embedding of a G_{bool} program into the chemtainer calculus remains exactly as in the case of *GOTO* programs.

5 From a Practical Embedding Towards a Higher Level Programming Language for Chemical Reaction Control

Thus far, we have shown how to map modified GOTO-programs to chemtainer systems and how those systems can simulate the computation of programs. While these embeddings reveal that the chemtainer calculus is indeed Turing complete, this does not come to a great surprise. However, our constructions are explicit; they constitute an algorithm that translates given programs to actual chemtainer systems that can be executed chemically. In that sense, we have outlined the construction of a very simple chemical compiler to capture the control-flow of a simplified programming language in a setup of artificial cellular matrices. Our current focus is now on adding proper chemical operations in the sense of a library to our framework and to continue improving our system to denote intended chemical reactions and products in a more declarative manner. In terms of semantics, we are working on a probabilistic interpretation to capture the nature of chemical reaction systems more accurately.

The embedding we have shown in this work is far away from what can be done in a laboratory. Nevertheless, we claim that our work has some practical implications. From a mathematical perspective, the presented embedding is a solid starting point for further developments. Solid because Turing completeness allows referring to a large body of well–established results. Adding further functions will not change the property of Turing completeness, but only facilitate the implementation (additional functions may be chosen with particular attention to chemical practicability). We aim at a bi-directional way of inspiration. Mathematical consideration may suggest specific functions to be of high usability (e.g., because they facilitate compilation). It is then a question for chemistry whether these functions can be implemented. Going in the other direction, biology provides us with sophisticated mechanisms, e.g. for the synthesis of branched oligosaccharides [18]. Given a mathematical framework, one may ask how to translate such evolved functions into a formal framework and to what extent they offer general tools.

One may even go a step further. In this work, we emphasize Turing completeness. Comparing our embedding to what one finds in biology may shed light on the role of Turing completeness. We don't assume biological systems to exhibit specific mathematical properties; it is, however, of interest to analyze in what respect biological process control differs from the ideal one has constructed in computer science.

Finally, we highlight the difference between procedural and declarative languages. The presented embedding follows the procedural paradigm. However, chemical kinetics is, by its very nature a prime example for a declarative language with a semantics that can be simulated by the Gillespie algorithm [19][20]. We claim that further progress towards the understanding of biological processes and chemical process control has to include a shift from the procedural to the declarative point of view in order to take account of the fundamental nature of chemistry.

References

1. Feinberg, M.: *Some recent results in chemical reaction network theory*, In Patterns and dynamics in reactive media, Springer, New York, NY, 1991.
2. Banzhaf, W., Yamamoto, L: *Artificial chemistries*, MIT Press, 2015.
3. Dittrich, P., Ziegler, . & Banzhaf, W.: *Artificial chemistries – a review*, Artificial life, 7(3), 2001.
4. Kim, Y., Kim, J. W., Kim, Z. & Kim, W. Y: *Efficient prediction of reaction paths through molecular graph and reaction network analysis*, Chemical science, 9(4), 2018.

5. Liu, Y., Sumpter, D. JT: *Mathematical modeling reveals spontaneous emergence of self-replication in chemical reaction systems*, Journal of Biological Chemistry, 293(49), 2018.
6. Cardelli, L., Češka, M., Fränzle, M., Kwiatkowska, M., Laurenti, L., Paoletti, N. & Whitby, M.: *Syntax-guided optimal synthesis for chemical reaction networks*, In International Conference on Computer Aided Verification, Springer, Cham, 2017.
7. Cardelli, L., Kwiatkowska, M. & Whitby, M.: *Chemical reaction network designs for asynchronous logic circuits*, Natural computing, 17(1), 2018.
8. Nardin, C., Widmer, J., Winterhalter, M. & Meier, W.: *Amphiphilic block copolymer nanocontainers as bioreactors*, The European Physical Journal E, 4(4), 2001.
9. Noireaux, V. & Libchaber, A.: *A vesicle bioreactor as a step toward an artificial cell assembly*, Proceedings of the National Academy of Sciences, 101(51), 2004.
10. Roodbeen, R. & Van Hest, J. C.: *Synthetic cells and organelles: compartmentalization strategies*, BioEssays, 31(12), 2009.
11. Baxani, D. K., Morgan, A. J., Jamieson, W. D., Allender, C. J., Barrow, D. A. & Castell, O. K.: *Bilayer Networks within a Hydrogel Shell: A Robust Chassis for Artificial Cells and a Platform for Membrane Studies*, Angewandte Chemie International Edition, 55(46), 2016.
12. Li, J. & Barrow, D. A.: *A new droplet-forming fluidic junction for the generation of highly compartmentalised capsules*, Lab on a Chip, 17(16), 2017.
13. Păun, G.: *Computing with membranes*, Journal of Computer and System Sciences, 61(1), 2000.
14. Regev, A., Panina, E.M., Silverman, W., Cardelli, L. & Shapiro, E.: *BioAmbients: an abstraction for biological compartments*, Theoretical Computer Science, 325(1), 2004.
15. Cardelli, L.: *Brane calculi: Interactions of biological membranes*, In Computational methods in systems biology, Springer, Cham, 2005.
16. Fellermann, H. & Cardelli, L.: *Programming chemistry in DNA-addressable bioreactors*, Journal of The Royal Society Interface, 11(99), 2014.
17. Schöning, U.: Theoretische Informatik - kurz gefasst, Spektrum Akademischer Verlag, 2003.
18. Weyland, M. S., Fellermann, H., Hadorn, M., Sorek, D., Lancet, D., Rasmussen, S., & Füchslin, R. M.: *The MATCHIT automaton: exploiting compartmentalization for the synthesis of branched polymers*, Computational and mathematical methods in medicine, 2013.
19. Gillespie, Daniel T.: *A General Method for Numerically Simulating the Stochastic Time Evolution of Coupled Chemical Reactions*, Journal of Computational Physics, 22(4): 403–434, 1976.
20. Gillespie, Daniel T.: *Exact Stochastic Simulation of Coupled Chemical Reactions*, The Journal of Physical Chemistry, 81(25), 1977.