Programming chemistry in DNA addressable bioreactors

Harold Fellermann

Newcastle University, School of Computing Science, King's Gate, Newcastle upon Tyne NE1 7RU, United Kingdom and

*Center for Fundamental Living Technology, University of Southern Denmark, Campusvej 55, 5230 Odense M, Denmark**

Luca Cardelli

Microsoft Research Cambridge, 21 Station Rd, Cambridge CB1 2FB, United Kingdom†

We present a formal calculus, termed *chemtainer calculus*, able to capture the complexity of compartmentalized reaction systems such as populations of possibly nested vesicular compartments. Compartments contain molecular cargo as well as surface markers in the form of DNA single strands. These markers serve as compartment addresses and allow for their targeted transport and fusion, thereby enabling reactions of previously separated chemicals. The overall system organization allows for the setup of programmable chemistry in microfluidic or other automated environments. We introduce a simple sequential programming language whose instructions are motivated by state-of-the-art microfluidic technology. Our approach integrates electronic control, chemical computing, and material production in a unified formal framework that is able to mimick the integrated computational and constructive capabilities of the subcellular matrix. We provide a non-deterministic semantics of our programming language that enables us to analytically derive the computational and constructive power of our machinary. This semantics is used to derive the sets of all constructable chemicals and supermolecular structures that emerge from different underlying instruction sets. Since our proofs are constructive, they can be utilized to automatically infer control programs for the construction of target structures from a limited set of resource molecules. Finally, we present an example of our framework from the area of oligosaccharide synthesis.

I. INTRODUCTION

Living systems are unique in that they integrate molecular recognition and information processing with material production on the molecular scale. The predominant locus of this integration is the cytoplasm, where a multitude of biochemical compounds is highly organized in vesicular compartments that co-locate reactants of desired reactions while separating those of undesired reactions. Surface markers on these compartments are used for vesicular trafficking, as well as vesicle budding and fusion, thereby allowing for the fine-tuned control of biochemical reaction cascades [1–3].

The desire to employ this complex molecular organization in next generation chemical synthesis has led to various studies on supermolecular compartments as nanoscale "bioreactors" [4–7]. Several pathways for vesicle fusion [8–10] have been suggested. In particular, Hadorn et al. employ short single stranded DNA tags for the specific interaction of various reaction compartments [11–13]. In the European Commission funded project MATCHIT [14–17], we are working in creating

*harold.fellermann@newcastle.ac.uk
†luca@microsoft.com

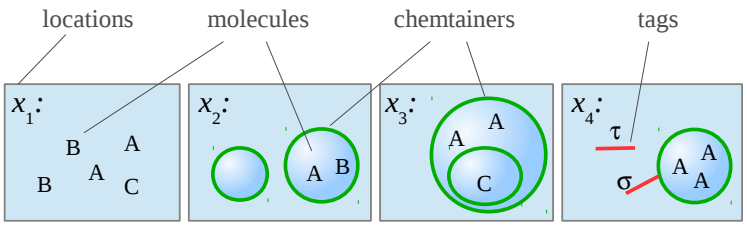

Figure 1. Example of a state of the artificial cellular matrix, where four locations ($x_1, x_2, x_3,$ and x_4) hold content. All other locations are empty. Location x_1 contains a number of molecules in solution ($2A+2B+C$), location x_2 contains two chemtainers (depicted as green circles), one of them containing the molecules A and B, location x_3 holds a chemtainer encapsulated within another chemtainer, and location x_4 holds a chemtainer with three A's being decorated with a tag σ (red line attached to the green circle), with a separate complementary tag τ in solution (free red line). Using the syntax defined in Section II A, this system state is described by the string $x_1 : 2A+2B+C \circ x_2 : \langle\!\langle\ \rangle\!\rangle + \langle\!\langle\ A + B\ \rangle\!\rangle \circ x_3 : \langle\!\langle\ 2A + \langle\!\langle\ C\ \rangle\!\rangle\ \rangle\!\rangle \circ x_4 : \tau + \sigma\langle\!\langle\ 3A\ \rangle\!\rangle$.

an artificial cellular matrix that seamlessly integrates information processing and material production in much the same way as its biological counterpart: The MATCHIT framework employs DNA addressable bioreactors (termed *chemtainers* in the following), to mimic the topological organization of the cytoplasm. DNA tags open up for DNA computing operations akin to the "key-lock" information processing mechanism found in biological protein-protein interactions. This form of molecular information processing is governed by autonomous chemical reaction kinetics and allows for tight integration of chemical production and information processing. In particular, we here employ the DNA *join-and-fork* gates to implement Boolean computation [18].

Whereas natural cells ultimately employ genomic information to regulate the resulting material production network, we envision programmable electronic control by embedding chemtainers into mechano-electronic microsystems [19, 20]. In such devices, dedicated microfluidic channels can be designed for vesicle generation [21], DNA tag insertion, tag and chemtainer trafficking [22], specific or unspecific fusion [23], vesicle rapture and encapsulation [24]. Possibly paired with real time feedback, this allows for control of chemical reaction cascades at the molecular level. This interplay of autonomous molecular computation and external electronic chemtainer manipulation enables the programmatic setup of complex reaction cascades that allow for automated chemical production of a desired target molecule from a limited set of chemical resources.

Here, we propose a formal calculus to describe system states and transitions in an abstract representation of the artificial cellular matrix. We call this the *chemtainer calculus*. In essence, the chemtainer calculus allows us to describe the organization and manipulation of chemical compounds down to the molecular level in possibly nested, addressable reaction compartments. Sets, or more precisely, multisets of such compartmentalized reagents are situated at locations, e.g., at positions of a microfluidic machine. Several calculi for compartmentalized reaction systems have been proposed previously [25–27]. Our syntax for nested and tagged compartments follows relatively closely the one from *brane calculi* [27]. Those calculi offer transitions for compartment transformations, such as fusion and splitting, as well as molecular reactions. Transitions are integral components of the system state and the calculus employs a process algebra to deduce the set of possible transitions from the current state. In the chemtainer calculus, we additionally define an explicit transition system that operates on states externally. This organization better reflects the difference between the chemical state transitions and external mechano-electronic control.

The calculus was designed for the architecture described in Ref. [20]. The tool chain for compilation of high level directives of the chemtainer calculus into eventual electrode configurations

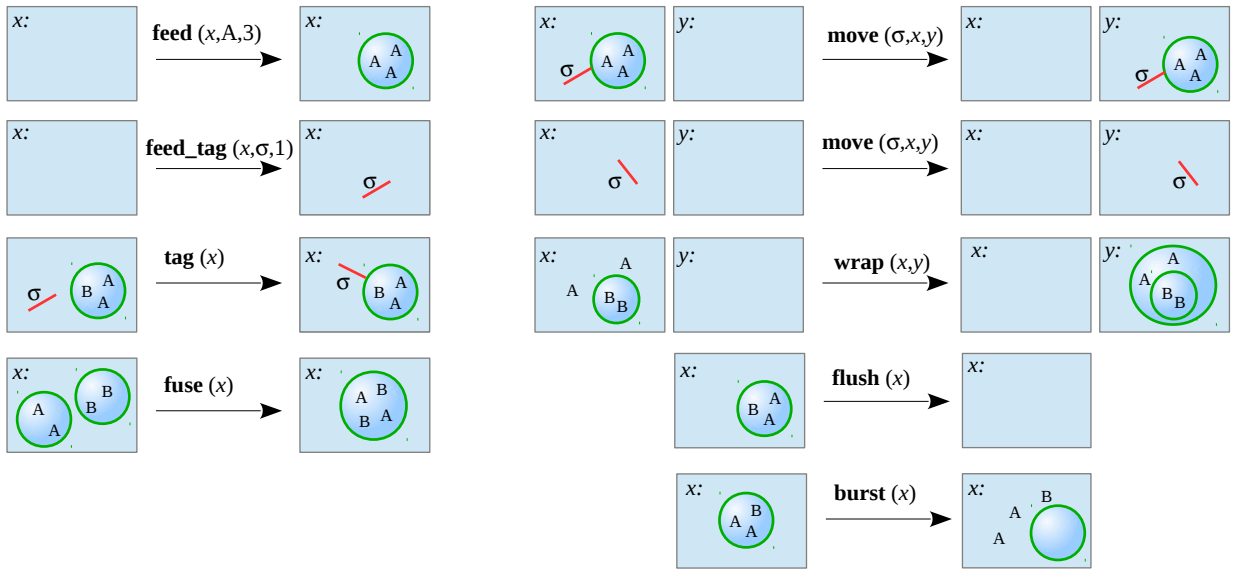

Figure 2. Schematics of programmable transitions and their instructions in the chemtainer calculus.

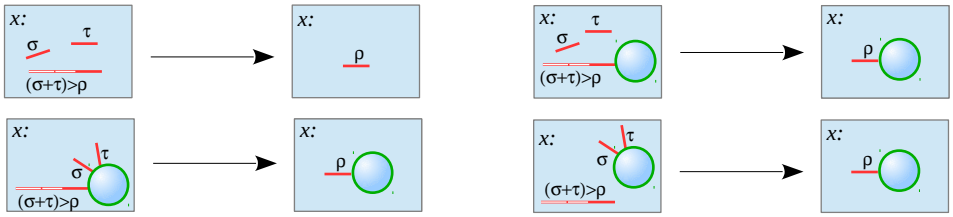

Figure 3. Transitions that encode gate reactions: A gate reacts with its respective input tags if the two are not separated by a chemtainer wall. Either the gate or its inputs might be attached to a chemtainer surface, in which case the output tags of the gate will equally be bound to the surface.

to control the mechano-electronic microfluidic hardware is presented in Ref. [28]. Yet, the general framework discussed here is not taylored toward one specific technology. For example, instead of microfluidic channel segments, locations could equally denote test tube arrays, or wells in a high throughput chip.

In this article, we first design a syntax that allows us to express the rather complex system states—or rather system *arrangements*—that we encounter in the artificial cellular matrix: located multisets of tagged and possibly nested chemtainers that carry cargo. Fig. 1 schematically depicts an example of a possible state. We then introduce transitions between states that model possible changes of the physical state. Some of these transitions, e.g. DNA hybridization, capture autonomous chemistry, whereas other transitions are thought to be induced by control operations of the artificial cellular matrix. We proceed by defining a simple programming language that consists of sequences of parametric instructions which induce transitions on the system state. In Section III, we discuss how the underlying instruction set of the chemtainer calculus affects the set of constructable objects, and we give our main result (Theorem 5) that a set of eight instructions is sufficient to construct any well-formed target state. Due to the fact that we use a non-deterministic semantics, this proofs only demonstrates the possibility, not the probability of

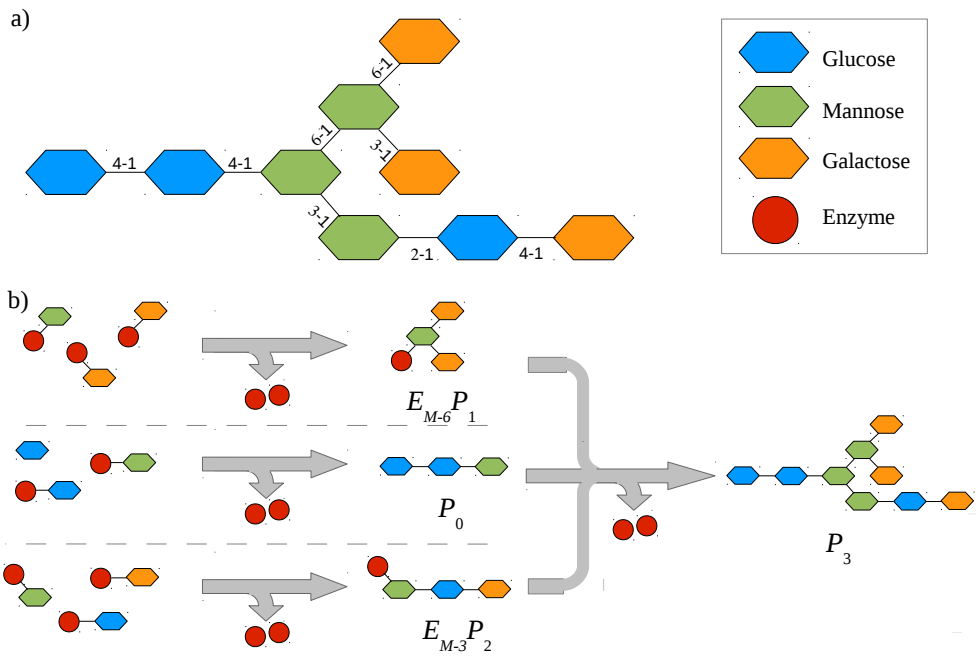

Figure 4. a) Example of an oligosaccharide target structure, consisting of specifically connected monomers of different types (hexagons). b) Reaction cascade to synthesize the target structure. Potential side reactions are avoided by encapsulating reactants into separate chemtainers (indicated by dashed lines).

creating a certain desired state. In Section IV, we apply our framework to the area of chemical manufacturing, where we present an algorithm for synthesis of branched oligomer structures by means of controlled chemtainer fusion and DNA computing.

II. THE CHEMTAINER CALCULUS

A. System state

Objects of the calculus are molecules, address tags, and chemtainers. Chemtainers represent compact microreactors, such as vesicles, oil droplets in water or water droplets in oil, DNA nanocages, etc. They can be decorated with address tags, and can hold chemical content and even other chemtainers within them. Here, we do not distinguish between different chemtainer types, but we can imagine a type system for chemtainers to specify their physical properties. All components of the system are situated at specified, discrete locations.

Our notion of space is rather simplistic: we assume a fixed set of locations; each component of the system state is situated at a certain location; we will introduce transitions that allow collocated components to interact (while objects at different locations may not interact), and we will introduce transitions that induce transport from one location to another by means of state transitions. Note that we currently do not introduce a specific topology on the set of locations (e.g., to move from one location to another, one might need to cross a third one), but such an extension is possible.

For countable index sets $J_{\mathcal{L}}$, $J_{\mathcal{M}}$, and $J_{\mathcal{T}}$, let $\mathcal{L} = \{x_i; i \in J_{\mathcal{L}}\}$ denote the set of locations, $\mathcal{M} = \{m_j; j \in J_{\mathcal{M}}\}$ some set of molecules, and $\mathcal{T} = \{s_k; k \in J_{\mathcal{T}}\}$ a set of DNA tags. We take $\mathcal{T} = \{\rho, \sigma, \tau, \ldots\}$ if tags are explicitly given. Note that $\mathcal{M}$ might intersect with $\mathcal{T}$ or not.

The following context-free tree grammar G_{S} for system states formalizes the above verbal in-

terpretation:

$$\text{global state} \qquad \mathsf{S} := \emptyset \mid \mathsf{S} \circ \mathsf{S} \mid x_i : \mathsf{P} \tag{1}$$

$$\text{local state} \qquad \mathsf{P} := 0 \mid \mathsf{P} + \mathsf{P} \mid \mathsf{q}^* \langle\!(\mathsf{P})\!\rangle \mid \mathsf{q} \mid m_j \tag{2}$$

$$\text{tag} \qquad \mathsf{q} := \mathsf{s} \mid \mathsf{s}^* \rhd \mathsf{s}^* \tag{3}$$

with the start symbol S. Here, the vertical broken bar is a metasymbol that indicates syntactic choice. We denote the empty state by the symbol $\emptyset$. The binary operator $x_i : \mathsf{P}$ denotes localization, where x_i is a location identifier; the binary operator $\mathsf{S} \circ \mathsf{S}$ denotes composition of locations, while $\mathsf{P}+\mathsf{P}$ denotes composition within locations; 0 denotes the empty local state; * is the Kleene operator and signifies no or arbitrarily many repetitions of its argument. We write

$$\mathsf{q}^* = \diamond \mid \mathsf{q} + \mathsf{q}^* \tag{4}$$

$$\mathsf{s}^* = \diamond \mid \mathsf{s} + \mathsf{s}^* \tag{5}$$

with the empty tag $\diamond$ to explicitly list tag and gate sets.

Following convention, we denote chemtainers by half moon parentheses $\mathsf{q}^* \langle\!(\mathsf{P})\!\rangle$ that enclose the chemtainer content with address tags associated to the left parenthesis [27, 29]. Finally, the relation $\mathsf{s}^* \rhd \mathsf{s}^*$ denotes DNA gates, which will be explained later. We write G_{S}, G_{P}, G_{q^*}, and G_{s^*} for the grammars with the start symbols S, P, q^*, and s^*, respectively.

To generate a state with the grammar G_{X}, $\mathsf{X} \in \{\mathsf{S}, \mathsf{P}, \mathsf{q}^*, \mathsf{s}^*\}$ being a nonterminal, one recursively applies the above production rules starting from X until the resulting state tree contains no more nonterminal symbols. The language $L(G_{\mathsf{X}})$ is the set of all possible states that can be generated from the start symbol X. In what follows, $S, S', S'' \in L(G_{\mathsf{S}})$, $P, P', P'', P_i \in L(G_{\mathsf{P}})$, $q, q', q_k \in L(G_{\mathsf{q}^*})$, and $s, s', s'', s_k \in L(G_{\mathsf{s}^*})$ denote arbitrary states of the respective languages.

Informally, we interpret states of $L(G_{\mathsf{S}})$ to signify the following: the global system state is a composition of local states, where each local state has a location identifier x_i and an associated local state description. The latter are compositions of molecules m_j, gates q_k, as well as chemtainers $q^* \langle\!(P)\!\rangle$ with content P and gate set q^*; gate sets, in turn, are compositions of gates q_k as well as individual tags s_k. See Fig. 1 for an example of a global state.

In order to conform with this interpretation, we introduce the following structural congruence relation (an equivalence relation with additional axioms that guarantee substitutivity of equals in context) over $L(G_{\mathsf{S}})$, $L(G_{\mathsf{P}})$, $L(G^*_{\mathsf{q}})$, and $L(G^*_{\mathsf{s}})$:

$$S \circ (S' \circ S'') \equiv (S \circ S') \circ S'' \tag{6}$$

$$S \circ S' \equiv S' \circ S \tag{7}$$

$$S \circ \emptyset \equiv S \tag{8}$$

$$P + (P' + P'') \equiv (P + P') + P'' \tag{9}$$

$$P + P' \equiv P' + P \tag{10}$$

$$P + 0 \equiv P \tag{11}$$

$$q_1 + (q_2 + q_3) \equiv (q_1 + q_2) + q_3 \tag{12}$$

$$q_1 + q_2 \equiv q_2 + q_1 \tag{13}$$

$$q + \diamond \equiv q \tag{14}$$

$$s_1 + (s_2 + s_3) \equiv (s_1 + s_2) + s_3 \tag{15}$$

$$s_1 + s_2 \equiv s_2 + s_1 \tag{16}$$

$$s + \diamond \equiv s \tag{17}$$

$$x_i : P \circ x_i : P' \equiv x_i : P + P' \tag{18}$$

$$\frac{S_1 \equiv S_2}{S \circ S_1 \equiv S \circ S_2} \quad (19) \qquad \frac{s_1 \equiv s_2}{s^* + s_1 \equiv s^* + s_2} \quad (23)$$

$$\frac{P_1 \equiv P_2}{x_i : P_1 \equiv x_i : P_2} \quad (20) \qquad \frac{s_1^* \equiv s_2^*}{s_1^* \rhd s^* \equiv s_2^* \rhd s^*} \quad (24)$$

$$\frac{P_1 \equiv P_2}{P + P_1 \equiv P + P_2} \quad (21) \qquad \frac{s_1^* \equiv s_2^*}{s^* \rhd s_1^* \equiv s^* \rhd s_2^*} \quad (25)$$

$$\frac{q_1 \equiv q_2}{q^* + q_1 \equiv q^* + q_2} \quad (22) \qquad \frac{P_1 \equiv P_2}{q^* \langle\!(P_1)\!\rangle \equiv q^* \langle\!(P_2)\!\rangle} \quad (26)$$

$$\frac{q_1^* \equiv q_2^*}{q_1^* \langle\!(P)\!\rangle \equiv q_2^* \langle\!(P)\!\rangle} \quad (27)$$

Thus, states that belong to the same equivalence class of $\equiv$ represent the same physical state, even though they might be syntactically different. Equations (6) through (18) induce monoidal structures on $(L(G_{\mathsf{S}}), \circ, \emptyset)$, $(L(G_{\mathsf{P}}), +, 0)$, $(L(G_{\mathsf{q}^*}), +, \diamond)$, and $(L(G_{\mathsf{s}^*}), +, \diamond)$, where ':' distributes over '+', and Equations (19) through (27) guarantee that we can substitute equals in any context.

We introduce some notational shortcuts. We write $\langle\!(P)\!\rangle := \diamond\langle\!(P)\!\rangle$, $q\langle\!(\ )\!\rangle := q\langle\!(0)\!\rangle$ and we introduce the notation

$$nP := \underbrace{P + \ldots + P}_{n \text{ times}} \quad (28)$$

which emphasizes the relation to mulitsets. We also allow ourselves to drop the explicit notation of empty locations by defining

$$x_i : 0 \equiv \emptyset. \quad (29)$$

With these definitions, the example state depicted in Fig. 1 can be written as

$$x_1 : 2A + 2B + C \circ x_2 : \langle\!(\ )\!\rangle + \langle\!(A + B)\!\rangle \circ x_3 : \langle\!(2A + \langle\!(C)\!\rangle)\!\rangle \circ x_4 : \tau + \sigma\langle\!(3A)\!\rangle. \quad (30)$$

B. Transitions

We are now ready to introduce a transition system that codifies the possible outcome of both autonomous chemical reactions, as well as externally induced operations that manipulate the system state. Autonomous reactions, in turn, are either arbitrary chemical reactions among molecules, which we refer to as *application chemistry*, or the working of DNA computing operations.

1. Application Chemistry

Reactions are just transitions of the form

$$P \longrightarrow P' \quad (31)$$

where $P = \sum_i \nu_i m_i$ and $P' = \sum_j \mu_j m_j$ are multisets of reactants of products with stoichiometric coefficients ν_i and μ_i.

We ensure that chemical reactions can occur among any co-located reactants that are not

separated by chemtainer walls:

$$\frac{P \longrightarrow P'}{P + P'' \longrightarrow P' + P''} \tag{32}$$

$$\frac{P \longrightarrow P'}{q^*(\!| P |\!) \longrightarrow q^*(\!| P' |\!)} \tag{33}$$

$$\frac{P \longrightarrow P'}{x_i : P \longrightarrow x_i : P'} \tag{34}$$

$$\frac{S \longrightarrow S'}{S \circ S'' \longrightarrow S' \circ S''} \tag{35}$$

We extend structural congruence to transitions by defining.

$$\frac{P \equiv P' \qquad P' \longrightarrow P'' \qquad P'' \equiv P'''}{P \longrightarrow P'''} \tag{36}$$

$$\frac{S \equiv S' \qquad S' \longrightarrow S'' \qquad S'' \equiv S'''}{S \longrightarrow S'''}\,. \tag{37}$$

In its ground form, the chemtainer calculus does not offer any transitions of the form (31). Instead, the user of the calculus is assumed to provide a set $\mathcal{R}$ of autonomous transitions as axioms.

Note that there is no particular need to restrict $\mathcal{M}$ to be finite. Our calculus can be applied without adaption to an infinite set of molecules, including polymers or branched structures—an example of which will be presented in Section IV. We also emphasize that we explicitly allow $\mathcal{M}$ and $\mathcal{T}$ to intersect. If they do, tags can occur in the reactant and product sides of chemical reactions, such that tags can be altered chemically.

2. DNA Gate Transitions

Here, DNA computation is implemented by *join and fork* gates [18] that irreversibly release a given set of output strands s_2^* once they have bound a set of inputs s_1^*, written $s_1^* \rhd s_2^*$. Note that $s_1^* \rhd s_2^*$ does not physically contain the strands s_1^*. Rather it means that the gate accepts those strands as input.

If a gate is co-located with all its input tags, it can release its output tag:

$$s_1^* \rhd s_2^* \; + \; s_1^* \longrightarrow s_2^* \tag{38}$$

We have to ensure that these transitions can occur between two co-located complementary tags in any context, unless they are separated by a chemtainer boundary. This is allowed by introducing the following inference rules:

$$\frac{q + q' \longrightarrow q''}{q + q'(\!| P |\!) \longrightarrow q''(\!| P |\!)} \tag{39}$$

$$\frac{q + q' \longrightarrow q''}{(q + q')(\!| P |\!) \longrightarrow q''(\!| P |\!)} \tag{40}$$

Examples of gate transitions are depicted in Fig. 3. Due to the inferences (32) through (37), its is guaranteed that DNA computing operations perform in any context.

3. Induced Transitions

We now introduce transitions that model the controlled manipulation of states through operations of the artificial cellular matrix. We introduce eight such operations, responsible for feeding and moving of chemtainers and tags, controlled fusion of chemtainers, encapsulation of material into chemtainers, chemtainer bursting, and flushing of content. Our exact transitions are motivated by the functionalities of the underlying microfluidics architecture [20], but they also serve as guideline how alternative transitions in other hardwares can be codified. The impact of the exact instruction set on the constructive capabilities of the resulting calculus will be discussed in Sections III.

Induced transitions are of the form

$$I : S \longrightarrow S$$

where I is a parameterized name. We first give the formal definition of these transitions (schematics are shown in Fig. 2) and will afterwards comment on their particular choice. In Sec. II C, we will introduce a small programming language based on these operations.

$$\mathbf{feed}(x, m_i, \nu): \quad \emptyset \longrightarrow x : \langle\!(\nu m_i)\!\rangle \tag{41}$$

$$\mathbf{feed\_tag}(x, s, \nu): \quad \emptyset \longrightarrow x : \nu s \tag{42}$$

$$\mathbf{move}(s, x, z): \quad x : (s + q^*)\langle\!(P)\!\rangle \longrightarrow z : (s + q^*)\langle\!(P)\!\rangle \tag{43}$$

$$x : s \longrightarrow z : s \tag{44}$$

$$\mathbf{tag}(x): \quad x : s + q^*\langle\!(P)\!\rangle \longrightarrow x : (s + q^*)\langle\!(P)\!\rangle \tag{45}$$

$$\mathbf{fuse}(x): \quad x : q_1^*\langle\!(P)\!\rangle + q_2^*\langle\!(P')\!\rangle \longrightarrow x : (q_1^* + q_2^*)\langle\!(P + P')\!\rangle \tag{46}$$

$$\mathbf{flush}(x): \quad x : P \longrightarrow \emptyset \tag{47}$$

$$\mathbf{wrap}(x, z): \quad x : P \longrightarrow z : \langle\!(P)\!\rangle \tag{48}$$

$$\mathbf{burst}(x): \quad x : q^*\langle\!(P)\!\rangle \longrightarrow x : q^*\langle\!(\ )\!\rangle + P \tag{49}$$

Transitions (41) through (49) operate strictly on the top level of the state, meaning that we do not allow, for example, tagging or fusion of chemtainers that resides within another chemtainer. To ensure this, we simply do not provide inference rules that would allow us to derive such transitions. However, the above transitions are allowed to operate in any context through the following inference:

$$\frac{I : S \longrightarrow S'}{I : S \circ \bar{S} \longrightarrow S' \circ \bar{S}} \tag{50}$$

We now comment on the individual choice of operations and their transitions. **feed** allows one to feed chemtainers that are equipped with a certain number $\nu \in \mathbb{N}$ of molecules of a single molecular species into a location. This is the only means to provide elements of $\mathcal{M}$. Similarly,

feed_tag allows one to feed a certain number $\nu \in \mathbb{N}$ of identical tags into a location. **tag** has been modeled to unspecifically attach some available tag to some container at a given location. No means are given to specify which tag or which chemtainer is transformed in this operation. The reason for this choice is that the actual linking of tags to a chemtainer involves linker molecules (biotin and streptavidin) that are common to all chemtainers and tags independent of the actual tag sequence or chemtainer type.

We provide a **flush** command to counteract the effect of feeding, where flushing simply removes content at a location. The operation is most easily implemented by literally flushing the material out of the system.

The **move** command allows one to move a specific tag, or a chemtainer decorated with a specific tag from one location to another. Notably, this allows for content separation: for example, the state $x : \sigma(\!(A)\!) + \tau(\!(B)\!)$ will transition into the state $x : \sigma(\!(A)\!) \circ y : \tau(\!(B)\!)$ in response to the operation $\mathbf{move}(\tau, x, y)$. In a certain interprestation, this can be understood as chemtainer docking, where the locations correspond to volumes of bulk fluid that themselve move along a microfluidic channel decorated with the complementary tag (here τ). Assuming that both chemtainers are initially at location x which is decorated with τ, the complementarily tagged chemtainer is allowed to temporarily hybridize to the channel wall while the bulk volume—and thus the locations—physically move along the channel. Non-matching content at the logical location x will remain at that location, although its physical position has changed. Matching content, however, will be at a different location, e.g. y and can be released into the corresponding bulk.

The operations **fuse** and **wrap** are more specific to the artificial cellular matrix and enable fusion and (vesicle) encapsulation. **fuse** induces unspecific fusion of co-located chemtainers, leading to mixing of both chemtainer surface and content. Besides unspecific fusion, tag specific fusion of vesicles has been achieved experimentally [30, 31]. A version of the chemtainer calculus featuring such tag specific fusion is presented in [32]. **wrap** enables encapsulation of material into vesicles, e.g. using interface transfer: it is assumed that the material at location x resides in aqueous phase which is first immersed into an oil phase where it forms surfactant coated water droplets—providing the inner layer of the future vesicle. Next, the water droplet passes a surfactant covered oil-water interface which provides the outer layer of the vesicle membrane. If the original state at x contains chemtainers, these are equally transfered into the interor of the new chemtainer, leading to nested chemtainers [12]. Microfluidic implementations of this protocol are presented in [33]. We will later analyze the power of this transition system with and without the **wrap** operation.

Complementing encapsulation, we provide a **burst** operation that releases the content of a chemtainer. We envision that **burst** does not fully disolve the chemtainer, but rather raptures the chemtainer wall temporarily, for example by means of a heat or salt shock. After bursting, the chemtainer will reconstitute itself and remain in the system, available for further processing.

In order to capture obvious implementation constraints of the physical machine, some of the operations are restricted to subsets of $\mathcal{L}$, e.g. $\mathbf{feed}(x)$ might require $x \in \mathcal{L}_{\mathbf{feed}} \subset \mathcal{L}$. It can be shown that the constructive power of the calculus is not affected by this limitation, as long as **move**, **tag**, and **burst** are allowed to operate on the entire set $\mathcal{L}$. Likewise, we could constrain the **move** command such that the target location has to be in a certain neighborhood set of the source location in order to capture, e.g., the channel topology of microfluidic devices.

We wish to clarify one point that might be counter-intuitive. Assume, for example, the state $x : 10\sigma$. What will be the action to the operation $\mathbf{move}(\sigma, x, y)$? Intuitively, we might expect that the system transitions into the state $y : 10\sigma$, meaning that all instances of σ have been moved. However, $x : 10\sigma$ is structurally equivalent to $x : 9\sigma \circ x : \sigma$ due to the distributive relation (18). To this state we can apply the transition $S \circ x : \sigma \longrightarrow S \circ y : \sigma$ equating $S = x : 9\sigma$, which results into $x : 9\sigma \circ y : \sigma$. Thus, **move**, and by similarity all other transitions, will only operate on *a single* instance. There is a simple way out however: in order to move all ten instances in the

example above, we can simply execute the move operation ten times in sequence. Our reason for this semantics is that it allows for consistent assignment of stochastic rates in a potential stochastic semantics [34, 35].

C. Programs

In the previous section we have informally given parametric names to transitions, such as $\mathbf{move}(s, x, y)$, $\mathbf{tag}(x)$, a.s.o. We next interpret parametric names as elemental operations of a programming language that can be used to operate the artificial cellular matrix. We do this by first defining another formal language, the formal language of chemtainer programs, and then expressing the semantics of this language by means of state transitions, the latter employing the language $L(G_{\mathsf{S}})$ of chemtainer states.

As with states, we define programs by a recursive grammar, which we initially keep as simple as possible:

$$\pi := \epsilon \mid I; \pi. \tag{51}$$

A program π can be either the empty program ϵ, or a (parametric) instruction followed by another program (the remainder), with instructions being separated by semicolons. Instructions I can be any of the formerly introduced parametric names $\mathbf{feed}(x_a, m_j, \nu)$, $\mathbf{feed\_tag}(x_a, q_k, \nu)$, $\mathbf{move}(s_k, x_a, x_b)$, $\mathbf{tag}(x_a)$, $\mathbf{wrap}(x_a, x_b)$, $\mathbf{fuse}(x_a)$, $\mathbf{burst}(x_a)$, and $\mathbf{flush}(x_a)$ with $x_a, x_b \in \mathcal{L}, m_j \in \mathcal{M}, q_k \in L(G_{\mathsf{q}^*})$ and $\nu \in \mathbb{N}$. Each instruction I has a set of associated transitions and we write $I : S \longrightarrow S'$ to denote any such associated transition (as was already done in the previous part).

Strictly, the concatenation operator ';' is only defined to concatenate single instructions with programs. In order to concatenate arbitrary programs π and π', we introduce an additional operator '**;**' for program concatenation through the following recursive definition:

$$\epsilon;\pi = \pi \tag{52}$$

$$(I; \pi);\pi' = I; (\pi;\pi') \tag{53}$$

Since both concatenation operators have different domains, it is clear from the context whether the concatenation refers to single instructions or programs. Therefore, we will drop the syntactic differentiation and use the symbol ';' for both operators.

A program π together with a system state S is expressed by the tuple $\langle \pi, S \rangle$ and we write $\langle \pi, S \rangle \longrightarrow S'$ to denote that the program π can transform the state S into the state S', called a *result* of π. Results are defined by the following structural operational semantics [36]:

$$\langle \epsilon, S \rangle \longrightarrow S \tag{54}$$

$$\frac{\langle \pi, S'' \rangle \longrightarrow S \qquad I : S' \longrightarrow S''}{\langle I; \pi, S' \rangle \longrightarrow S} \tag{55}$$

$$\frac{\langle \pi, S' \rangle \longrightarrow S}{\langle I; \pi, S' \rangle \longrightarrow S} \tag{56}$$

In words, given a program that starts with instruction I with some associated transition from S' to S'', the second rule allows us to transform the system into S'', thereby reducing the original program to whatever remains after execution of I. If program execution encounters an instruction with no associated transition that is applicable to the current state S' (e.g. **tag** is asked to operate

on an empty cell), the third rule allows us to skip the instruction without modifying the system state. This latter rule primarily addresses erroneous conditions and ensures that a program does not get stuck, simply because the respective molecules are not present at some point of the execution. Using non-deterministic semantics, rule (56) might even be used when rule (55) is applicable, thereby skipping instructions of the program sequence. This could be remedied with a stochastic semantics that assigns an arbitrarily small transition rate to rule (56).

We can now distinguish between autonomous and induced transitions, by extending the semantics with the rules

$$\frac{S \longrightarrow S' \qquad \langle \pi, S' \rangle \longrightarrow S''}{\langle \pi, S \rangle \longrightarrow S''} \tag{57}$$

$$\frac{\langle \pi, S \rangle \longrightarrow S' \qquad S' \longrightarrow S''}{\langle \pi, S \rangle \longrightarrow S''} \tag{58}$$

where $S \longrightarrow S'$ and $S' \longrightarrow S''$ is an autonomous transition inferred through (35). These rules allow autonomous transitions to occur at any time during program execution without affecting the program.

We extend structural congruence to program application using the inference rule

$$\frac{S \equiv S'' \qquad \langle \pi, S \rangle \longrightarrow S' \qquad S' \equiv S'''}{\langle \pi, S'' \rangle \longrightarrow S'''}. \tag{59}$$

The following two lemmas, proven in the appendix, allow for uncomplicated composition of instructions.

Lemma 1 *Program application is not context sensitive: if the application of program π on the initial state S can lead to the result S', the application on the composition $S \circ \bar{S}$ can lead to the result $S' \circ \bar{S}$, containing S' as a substate. $\bar{S}$ is an invariant of π:*

$$\langle \pi, S \rangle \longrightarrow S' \quad \Longrightarrow \quad \left\langle \pi, S \circ \bar{S} \right\rangle \longrightarrow S' \circ \bar{S}. \tag{60}$$

Lemma 2 *Programs can be concatenated and the start configuration of the second program will be the result of the first program:*

$$\left\langle \pi', S \right\rangle \longrightarrow S' \quad \wedge \quad \left\langle \pi'', S' \right\rangle \longrightarrow S'' \quad \Longrightarrow \quad \left\langle \pi'; \pi'', S \right\rangle \longrightarrow S'' \tag{61}$$

D. Extension: control flow directives and parallel execution

We could easily define control flow directives known from standard programming languages, such as branched execution and loops. This is particularly interesting in the context of feedback control. Let

$$p : S \longrightarrow \{\text{True}, \text{False}\} \tag{62}$$

be a conditional over the system state. For example, f could measure whether a fluorescent signal at a certain location exceeds a certain threshold. We could then extend our grammar to

$$\pi := \epsilon \mid I; \pi \mid \textbf{while } f(S) \textbf{ do } \pi; \textbf{ endwhile} \tag{63}$$

to allow for conditional loops, where the semantics of the **while** statement are given by

$$\frac{\langle \pi, S\rangle \longrightarrow S' \qquad \langle \textbf{while } p \textbf{ do } \pi;\ \textbf{endwhile}, S'\rangle \longrightarrow S'' \qquad p = \text{True}}{\langle \textbf{while } p \textbf{ do } \pi;\ \textbf{endwhile}, S\rangle \longrightarrow S''} \tag{64}$$

$$\frac{p = \text{False}}{\langle \textbf{while } p \textbf{ do } \pi;\ \textbf{endwhile}, S\rangle \longrightarrow S} \tag{65}$$

As evidenced with these definitions, there is nothing particular about control flow directives in the chemtainer calculus compared to other imperative languages. In order to introduce more elaborate control flows, the reader can simply apply standard text book procedures.

Similarly, it is straightforward to define the semantics for a language of communicating sequential processes [37, 38] that parallelizes the production process:

$$\frac{\langle \pi', S'\rangle \longrightarrow \bar{S}' \qquad \langle \pi'', S''\rangle \longrightarrow \bar{S}''}{\langle \pi' | \pi'', S' \circ S''\rangle \longrightarrow \bar{S}' \circ \bar{S}''} \tag{66}$$

E. Example

To make the above definitions more familiar, we employ the chemtainer calculus in order to decorate chemtainers with addresses and use those to mix the contents of two chemtainers.

The left column shows which parametric instruction is executed and the right column shows the effect of its associated transition. Read from top to bottom, the left column therefore simply gives a program that can construct some desired state denoted at the right.

$$\begin{array}{ll} & \emptyset \\ \textbf{feed}(x, P) & x : \langle\!\langle\ P\ \rangle\!\rangle \\ \textbf{feed\_tag}(y, \sigma) & x : \langle\!\langle\ P\ \rangle\!\rangle \circ y : \sigma \\ \textbf{move}(\sigma, y, x) & x : \sigma + \langle\!\langle\ P\ \rangle\!\rangle \\ \textbf{tag}(x) & x : \sigma\langle\!\langle\ P\ \rangle\!\rangle \end{array} \tag{67}$$

To use this sequence later on in programs, we define it as the parametric macro $\textbf{resource}(x, \sigma, P)$. Now, we use this macro to create two chemtainers and fuse them:

$$\begin{array}{ll} & \emptyset \\ \textbf{resource}(x, \sigma, P) & x : \sigma\langle\!\langle\ P\ \rangle\!\rangle \\ \textbf{resource}(y, \rho, Q) & x : \sigma\langle\!\langle\ P\ \rangle\!\rangle \circ y : \rho\langle\!\langle\ Q\ \rangle\!\rangle \\ \textbf{move}(\rho, y, x) & x : \sigma\langle\!\langle\ P\ \rangle\!\rangle + \rho\langle\!\langle\ Q\ \rangle\!\rangle \\ \textbf{fuse}(x) & x : (\sigma + \rho)\langle\!\langle\ P + Q\ \rangle\!\rangle \end{array} \tag{68}$$

Keep in mind that the right column only shows a possible response of the system state to the program on the left, not the necessary response. In general, because of the ambiguity of transitions, the system could have transitioned into alternative states in response to the same program.

III. INSTRUCTION SETS AND THEIR CONSTRUCTIVE POWER

The instructions given in Eqs. (41) through (49) serve as examples of possible operations, rather than the final specification of an artificial cellular matrix. Potential real-world implementations

might involve only a subset of the operations mentioned, or might enable other means of chemtainer manipulation. It then becomes interesting to determine whether different instruction sets are equivalent in what they can construct or whether the set of constructable states differs.

In order to approach the problem formally, we define the *constructive closure* $\omega^+(\Pi_\mathsf{x}) \subset L(G_\mathsf{S})$ of the language Π_x as the set of states that can be constructed from the empty state through some program $\pi \in \Pi_\mathsf{x}$. Formally:

$$\omega^+(\Pi_\mathsf{x}) := \{S \in L(G_\mathsf{S}) : \exists \pi \in \Pi_\mathsf{x} \wedge \langle \pi, \emptyset \rangle \longrightarrow S\} \tag{69}$$

Similarly, we define the *reset closure* $\omega^-(\Pi_\mathsf{x}) \subset L(G_\mathsf{S})$ as the set of states that can be transformed back into the empty set. Formally:

$$\omega^-(\Pi_\mathsf{x}) := \left\{S \in \omega^+(\Pi_\mathsf{x}) : \exists \pi \in \Pi_\mathsf{x} \wedge \langle \pi, S \rangle \longrightarrow \emptyset\right\} \tag{70}$$

Of primary interest is of course the case $\omega^- = \omega^+$ where the machine can be reset from any state. In this case, one can formally transform any initial state into any target state, thereby programming sequences of states: if $\langle \pi, S\rangle \to \emptyset$ and $\langle \pi', \emptyset\rangle \to S'$, the concatenation $\pi;\pi'$ will transform S into S': $\langle \pi;\pi', S\rangle \to S'$. This result, however, is mostly of theoretical interest, as an interim reset of the machine might not be desired when programming such state sequences. In a practical application, one would define a distance measure between states and ensure that the transition path length of programs is minimized, thereby preferring for example a single **move** instruction over a sequence of **flush;feed_tag;move** instructions with equal outcome.

We consider the following three incremental instruction sets:

$$I_{\mathbf{min}} = \{\mathbf{feed}, \mathbf{feed\_tag}, \mathbf{move}, \mathbf{tag}, \mathbf{fuse}, \mathbf{flush}\} \tag{71}$$

$$I_{\mathbf{wrap}} = I_{\mathbf{min}} \cup \{\mathbf{wrap}\} \tag{72}$$

$$I_{\mathbf{burst}} = I_{\mathbf{wrap}} \cup \{\mathbf{burst}\} \tag{73}$$

Let $\Pi_{\mathbf{min}}$, $\Pi_{\mathbf{wrap}}$, and $\Pi_{\mathbf{burst}}$ be the sets of programs over the respective instruction set. We will show the following relations between the corresponding constructive and reset closures:

$$\begin{array}{ccccccc} \omega^+(\Pi_{\mathbf{min}}) & \subset & \omega^+(\Pi_{\mathbf{wrap}}) & \subset & \omega^+(\Pi_{\mathbf{burst}}) & = & L(G_\mathsf{S}) \\ \| & & \| & & \cup\| & & \\ \omega^-(\Pi_{\mathbf{min}}) & \subset & \omega^-(\Pi_{\mathbf{wrap}}) & \subset & \omega^-(\Pi_{\mathbf{burst}}) & \subseteq & L(G_\mathsf{S}). \end{array} \tag{74}$$

If and only if all locations are flushable, i.e. if $\mathcal{L}_{\mathbf{flush}} = \mathcal{L}$, the right-most chain of relations simplifies to

$$\omega^+(\Pi_{\mathbf{burst}}) = \omega^-(\Pi_{\mathbf{burst}}) = L(G_\mathsf{S}). \tag{75}$$

Diagram (74) can be decomposed into several theorems, for which we need to define the *nesting level* $d : L(G_\mathsf{S}) \to \mathbb{N}$ of a state S or local state P as the following:

$$d(\emptyset) = 0 \tag{76}$$

$$d(0) = 0 \tag{77}$$

$$d(m_j) = 0 \tag{78}$$

$$d(q^*) = 1 \tag{79}$$

$$d(x_i : P) = d(P) \tag{80}$$

$$d(S \circ S') = \max\{d(S), d(S')\} \tag{81}$$

$$d(P + P') = \max\{d(P), d(P')\} \tag{82}$$

$$d(q\langle\!\langle\, P \,\rangle\!\rangle) = 1 + d(P) \tag{83}$$

With this definition, we can state the relations of diagram (74) more precisely:

Theorem 3 *The constructive closure of* $\Pi_{\mathbf{min}}$ *is the set of states that do not contain nested chemtainers nor free floating molecules other than tags:*

$$\omega^+(\Pi_{\mathbf{min}}) = \left\{S \in L(G_\mathsf{S}) : d(S) \leq 1 \wedge S \not\equiv S' \circ x_i : m_j \text{ for } m_j \in \mathcal{M}\backslash\mathcal{T}\right\} \tag{84}$$

Theorem 4 *The constructive closure of* $\Pi_{\mathbf{wrap}}$ *is the set of states that do not contain free floating molecules other than tags:*

$$\omega^+(\Pi_{\mathbf{wrap}}) = \left\{S \in L(G_{\mathsf{S}}) : S \not\equiv S' \circ x_i : m_j \text{ for } m_j \in \mathcal{M}\backslash\mathcal{T}\right\} \tag{85}$$

Theorem 5 *The constructive closure of* $\Pi_{\mathbf{burst}}$ *is the entire set of states :*

$$\omega^+(\Pi_{\mathbf{burst}}) = L(G_{\mathsf{S}}) \tag{86}$$

Theorem 6 *The reset closure of all instruction sets is limited to states that do not contain free floating molecules other than tags in non-flushable locations:*

$$\omega^-(\Pi_{\mathsf{x}}) \subset \left\{S \in L(G_{\mathsf{S}}) : S \not\equiv S' \circ x_i : m_j \text{ for } x_i \notin \mathcal{L}_{\mathbf{flush}} \cup \mathcal{L}_{\mathbf{wrap\_in}}, m_j \in \mathcal{M}\backslash\mathcal{T}\right\} \quad \forall \mathsf{x} \in \{\mathbf{min}, \mathbf{wrap}, \mathbf{burst}\} \tag{87}$$

Proofs for all theorems are given in the appendix. Note that these proofs are constructive: for any given target structure $S \in \omega^+(\Pi_{\mathsf{x}})$ we can *automatically* generate a program that will construct S. This observation forms the core of a compiler that is able to generate a sequence of transitions for building a given target structure.

The above theorems assume that the entire set of molecules $\mathcal{M}$ can be fed as resources. We now address the case where only a subset $\mathcal{M}_{\mathrm{R}} \subset \mathcal{M}$ of compounds can be directly provided by **feed** and where chemical reactions are employed to produce compounds outside $\mathcal{M}_{\mathrm{R}}$.

For some reaction $\nu_1 m_1 + \nu_2 m_2 \longrightarrow \nu_3 m_3 + \nu_4 m_4$ with $m_1, m_2 \in \mathcal{M}_{\mathrm{R}}$, we can employ the program

$$\begin{array}{ll} & \emptyset \\ \mathbf{resource}(x, \psi, \nu_1 m_1) & x : \psi\langle\!\langle\ \nu_1 m_1\ \rangle\!\rangle \\ \mathbf{resource}(y, \rho, \nu_2 m_2) & x : \psi\langle\!\langle\ \nu_1 m_1\ \rangle\!\rangle \circ y : \rho\langle\!\langle\ \nu_2 m_2\ \rangle\!\rangle \\ \mathbf{move}(\rho, y, x) & x : \psi\langle\!\langle\ \nu_1 m_1\ \rangle\!\rangle + \rho\langle\!\langle\ \nu_2 m_2\ \rangle\!\rangle \\ \mathbf{fuse}(x) & x : (\psi + \rho)\langle\!\langle\ \nu_1 m_1 + \nu_2 m_2\ \rangle\!\rangle \\ & x : (\psi + \rho)\langle\!\langle\ \nu_3 m_3 + \nu_4 m_4\ \rangle\!\rangle \end{array} \tag{88}$$

to initiate the chemical production of $P' = \nu_3 m_3 + \nu_4 m_4$, where m_3, m_4 are not necessarily elements of $\mathcal{M}_{\mathrm{R}}$. Applying this procedure repeatedly allows us to obtain successively bigger subsets of $\mathcal{M}$. However, without means for content separation, the set of producible states will be constrained by the stoichiometries of the reactions. If the artificial cellular matrix would offer content separation, e.g. by means of electrophoresis [39], we could encode this with the induced transition

$$\mathbf{separate}(x, y): \quad x : (q + q')\langle\!\langle\ P + P'\ \rangle\!\rangle \longrightarrow y : q\langle\!\langle\ P\ \rangle\!\rangle + q'\langle\!\langle\ P'\ \rangle\!\rangle. \tag{89}$$

where the exact partitioning of chemtainer content into P and P' would depend on its electrophoretic mobility. Continuing the above program, we could now derive

$$\begin{array}{ll} & x : (\psi + \rho)\langle\!\langle\ \nu_3 m_3 + \nu_4 m_4\ \rangle\!\rangle \\ \mathbf{separate}(x, y) & y : \psi\langle\!\langle\ \nu_3 m_3\ \rangle\!\rangle + \rho\langle\!\langle\ \nu_4 m_4\ \rangle\!\rangle \end{array} \tag{90}$$

The tagged chemtainers are ready to be used as input for subsequent reactions. This recovers the constructive power of the chemtainer calculus, provided that all elements of $\mathcal{M}$ can be constructed by some chain of reactions. However, in order to maintain a well-defined mapping from DNA tags to chemtainer content, the **separate** operation would need to assert the correct redistribution of surface tags along with the content separation such that products can be unambiguously identified by their tags.

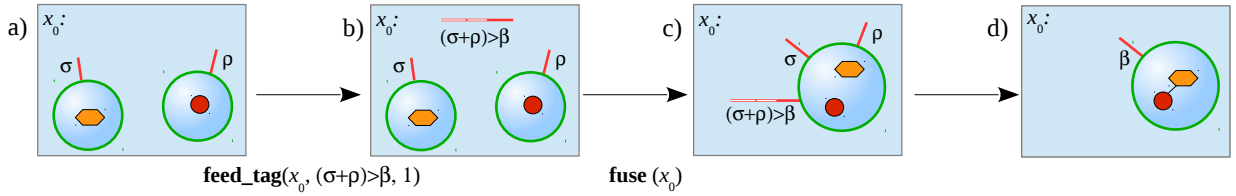

Figure 5. Steps for preparing enzymatically activated saccharide monomers. a) Chemtainers with activated enzymes and monomers are fed to the same location. b) A DNA gate is fed to the same location. c) Fusion of chemtainers co-locates the reactants in a single chemtainer, to which the provided DNA gate attaches. d) The reaction takes place inside the chemtainer while the result of chemtainer fusion is reflected by the successful transition of the DNA gate.

IV. PROGRAMMABLE REACTION CASCADES

Here, we demonstrate how our calculus (even with the minimal instruction set $I_{\mathbf{min}}$) can be used to integrate chemical production with molecular computation in order to control programmable reaction cascades. This example is motivated by and closely mimicks the synthesis of oligosaccharides in the Golgi apparatus [1, 2]. Oligosaccharides are branched, heterogeneous polymers composed of typically five to ten individual sugar monomers such as mannose, galactose, and glucose. This diverse class of biochemicals is involved in various physiological processes pertaining e.g to cell-cell recognition, intra- and intercellular trafficking, and metabolic modulation [40]. However, their combinatorial richness poses a challenge for chemical oligosaccharide synthesis based on conventional chemical manufacturing techniques [41].

Biological oligosaccharide synthesis proceeds in the Golgi apparatus by adding individual monomers one unit at a time to specific binding sites of the growing oligomer. Monomers are attached to enzymes that promote the specific binding reactions:

$$P_i + \mathrm{E}_j^* M_i \longrightarrow P_{i+1} + \mathrm{E}_j. \tag{91}$$

Here, P_i denotes an intermediate oligomer, to which monomer M_i is added at the site j. If binding sites are unique, a given enzyme-monomer complex $\mathrm{E}_j^* M_i$ contains all the information required to build the specific product P_{i+1} from P_i. Prior to these polymerization steps, monomers have to be attached to the respective enzymes:

$$M_i + \mathrm{E}_j^* \longrightarrow \mathrm{E}_j^* M_i. \tag{92}$$

Chemical one-pot synthesis of a given target structure is challenging, because repetition of bindings sites in the oligomer structure can lead to undesired side products. The number of potential side products can be controlled, however, by forcing some reaction steps to occur sequentially while others are allowed to proceed in parallel [42]. Weyland et al. [43] present an algorithm that identifies such optimal reaction cascades. For example, assume that the structure shown in Fig. 4 can be produced with the reaction cascade

$$\mathrm{Gal} + \mathrm{E}^*_{\text{Gal-4}}\mathrm{Gal} + \mathrm{E}^*_{\text{Gal-4}}\mathrm{Man} \longrightarrow P_0 + 2\,\mathrm{E}_{\text{Gal-4}} \tag{93}$$

$$\mathrm{E}^*_{\text{Man-6}}\mathrm{Man} + \mathrm{E}^*_{\text{Man-6}}\mathrm{Glc} + \mathrm{E}^*_{\text{Man-3}}\mathrm{Glc} \longrightarrow P_1\mathrm{E}^*_{\text{Man-6}} + \mathrm{E}_{\text{Man-6}} + \mathrm{E}_{\text{Man-3}} \tag{94}$$

$$\mathrm{E}^*_{\text{Man-3}}\mathrm{Man} + \mathrm{E}^*_{\text{Man-2}}\mathrm{Gal} + \mathrm{E}^*_{\text{Gal-4}}\mathrm{Glc} \longrightarrow P_2\mathrm{E}^*_{\text{Man-3}} + \mathrm{E}_{\text{Man-2}} + \mathrm{E}_{\text{Gal-4}} \tag{95}$$

$$P_0 + \mathrm{E}^*_{\text{Man-6}}P_1 + \mathrm{E}^*_{\text{Man-3}}P_2 \longrightarrow P_3 + \mathrm{E}_{\text{Man-6}} + \mathrm{E}_{\text{Man-3}} \tag{96}$$

where it has to be ensured that reactions (93) through (95) occur in isolation and prior to reaction (96).

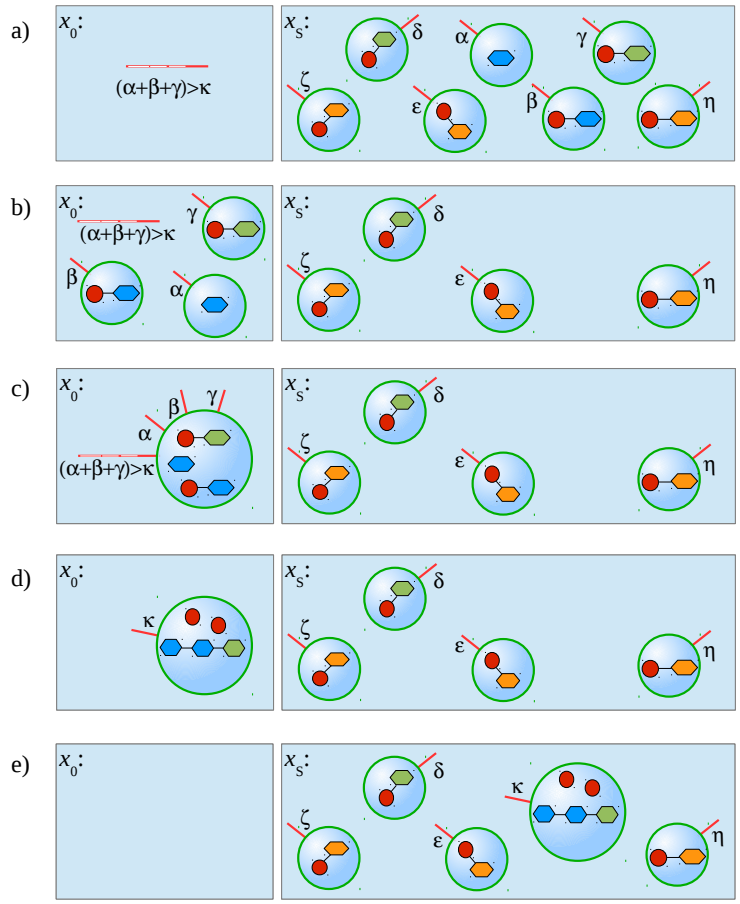

Figure 6. Steps in the programmed synthesis of compound P_0 from Fig. 4. a) Initial system state, where resource chemtainers are provided at the storage location x_S and the DNA gate $(\alpha+\beta+\gamma) \rhd \kappa$ is provided at the operation location x_0. b) System state after chemtainers labelled α, β, and γ are moved to x_0. c) System state after fusion has been initiated in x_0 and the tag has been bound. d) Chemical reactions and gate transitions occurr autonomously at x_0. e) The newly created chemtainer labelled κ with product P_0 is transferred back to x_S.

Our strategy here is to employ the chemtainers in order to control the encounter of reactants and hence the order of reactions by encapsulating reactants within chemtainers. Fusion of chemtainers co-locates desired reactants and triggers their polymerization. Simultaneously, DNA gate computation on the chemtainer surface in order will reflect the change of chemtainer content after reaction. The same DNA computation can ensure that reactions are started if and only if other reactions have been performed beforehand. We use one location x_0 for processing and one location x_S for storing intermediate products.

We start by preparing chemtainers with enzyme complexes using the program given in Equation (88). For example,

$$\mathbf{resource}(x_0, \sigma, \mathrm{Gal}); \mathbf{resource}(x_0, \rho, \mathrm{E}^*_{\text{Gal-4}}); \mathbf{feed\_tag}(x_0, (\sigma+\rho) \rhd \beta); \\ \mathbf{fuse}(x_0); \mathbf{move}(\beta, x_0, x_S) \quad (97)$$

creates the state $x_S : \beta\langle\!\langle\, \mathrm{E}^*_{\text{Gal-4}}\mathrm{Gal} \,\rangle\!\rangle$. See Fig. 5 for a graphical representation of these steps. We

use this algorithm repeatedly to set up the machine in the following state:

$$x_S : \alpha\langle\!(\ \mathrm{Gal}\ )\!\rangle\ +\ \beta\langle\!(\ \mathrm{E}^*_{\text{Gal-4}}\mathrm{Gal}\ )\!\rangle\ +\ \gamma\langle\!(\ \mathrm{E}^*_{\text{Gal-4}}\mathrm{Man}\ )\!\rangle + \delta\langle\!(\ \mathrm{E}^*_{\text{Man-6}}\mathrm{Man}\ )\!\rangle\ +\ \epsilon\langle\!(\ \mathrm{E}^*_{\text{Man-6}}\mathrm{Glc}\ )\!\rangle\ +\ \zeta\langle\!(\ \mathrm{E}^*_{\text{Man-3}}\mathrm{Glc}\ )\!\rangle + \eta\langle\!(\ \mathrm{E}^*_{\text{Man-3}}\mathrm{Man}\ )\!\rangle\ +\ \theta\langle\!(\ \mathrm{E}^*_{\text{Man-2}}\mathrm{Gal}\ )\!\rangle\ +\ \iota\langle\!(\ \mathrm{E}^*_{\text{Gal-4}}\mathrm{Glc}\ )\!\rangle \quad (98)$$

With this mapping from chemical compounds to DNA tags, we translate the reaction cascade (93) and (96) into a set of DNA gates:

$$(\alpha+\beta+\gamma) \rhd \kappa \tag{99}$$
$$(\delta+\epsilon+\zeta) \rhd \lambda \tag{100}$$
$$(\eta+\theta+\iota) \rhd \mu \tag{101}$$
$$(\kappa+\lambda+\mu) \rhd \omega \tag{102}$$

To carry out reaction (93), we feed the gate (99) at location x_0. We then progressively move chemtainers from x_A, x_B, x_C to x_0, initiate their fusion, and transport the intermediate product P_0 back to x_S (see Fig. 6 for a schematic representation):

$$\begin{array}{ll}
 & x_0 : (\alpha+\beta+\gamma) \rhd \kappa\ \circ\ x_S : \alpha\langle\!(\ \mathrm{Gal}\ )\!\rangle + \beta\langle\!(\ \mathrm{E}^*_{\text{Gal-4}}\mathrm{Gal}\ )\!\rangle + \gamma\langle\!(\ \mathrm{E}^*_{\text{Gal-4}}\mathrm{Man}\ )\!\rangle \\
\mathbf{move}(\alpha, x_S, x_0) & x_0 : (\alpha+\beta+\gamma) \rhd \kappa + \alpha\langle\!(\ \mathrm{Gal}\ )\!\rangle\ \circ\ x_S : \beta\langle\!(\ \mathrm{E}^*_{\text{Gal-4}}\mathrm{Gal}\ )\!\rangle + \gamma\langle\!(\ \mathrm{E}^*_{\text{Gal-4}}\mathrm{Man}\ )\!\rangle \\
\mathbf{move}(\beta, x_S, x_0) & x_0 : (\alpha+\beta+\gamma) \rhd \kappa + \alpha\langle\!(\ \mathrm{Gal}\ )\!\rangle + \beta\langle\!(\ \mathrm{E}^*_{\text{Gal-4}}\mathrm{Gal}\ )\!\rangle\ \circ\ x_S : \gamma\langle\!(\ \mathrm{E}^*_{\text{Gal-4}}\mathrm{Man}\ )\!\rangle \\
\mathbf{move}(\gamma, x_S, x_0) & x_0 : (\alpha+\beta+\gamma) \rhd \kappa + \alpha\langle\!(\ \mathrm{Gal}\ )\!\rangle + \beta\langle\!(\ \mathrm{E}^*_{\text{Gal-4}}\mathrm{Gal}\ )\!\rangle + \gamma\langle\!(\ \mathrm{E}^*_{\text{Gal-4}}\mathrm{Man}\ )\!\rangle \\
\mathbf{tag}(x_0) & x_0 : (\alpha + (\alpha+\beta+\gamma) \rhd \kappa)\langle\!(\ \mathrm{Gal}\ )\!\rangle + \beta\langle\!(\ \mathrm{E}^*_{\text{Gal-4}}\mathrm{Gal}\ )\!\rangle + \gamma\langle\!(\ \mathrm{E}^*_{\text{Gal-4}}\mathrm{Man}\ )\!\rangle \\
\mathbf{fuse}(x_0) & x_0 : (\alpha+\beta + (\alpha+\beta+\gamma) \rhd \kappa)\langle\!(\ \mathrm{Gal} + \mathrm{E}^*_{\text{Gal-4}}\mathrm{Gal}\ )\!\rangle + \gamma\langle\!(\ \mathrm{E}^*_{\text{Gal-4}}\mathrm{Man}\ )\!\rangle \\
\mathbf{fuse}(x_0) & x_0 : (\alpha+\beta+\gamma + (\alpha+\beta+\gamma) \rhd \kappa)\langle\!(\ \mathrm{Gal} + \mathrm{E}^*_{\text{Gal-4}}\mathrm{Gal} + \mathrm{E}^*_{\text{Gal-4}}\mathrm{Man}\ )\!\rangle \\
 & x_0 : \kappa\langle\!(\ P_0 + 2\,\mathrm{E}_{\text{Gal-4}}\ )\!\rangle \\
\mathbf{move}(\kappa, x_0, x_S) & x_S : \kappa\langle\!(\ P_0 + 2\,\mathrm{E}_{\text{Gal-4}}\ )\!\rangle \\
\mathbf{flush}(x_0) & x_S : \kappa\langle\!(\ P_0 + 2\,\mathrm{E}_{\text{Gal-4}}\ )\!\rangle
\end{array} \tag{103}$$

The **flush** instruction cleans the processing location in the case that the execution of an instruction got skipped because of rule (56). This prevents the subsequent fuse operations from operating on chemtainers that where intended to be kept separate.

Repeating the above algorithm with gate (100) and tags δ, ϵ, ζ as well as gate (101) and tags η, θ, ι produces chemtainers with the intermediate products P_0, P_1, and P_2 taged with κ, λ, and μ, respectively. Eventually, the above algorithm with gate (102) and tags κ, λ, μ produces the chemtainer $\omega\langle\!(\ P_3 + \ldots\ )\!\rangle$ which can readily be moved to some output location.

Note that the above program operates over a finte set of resources, tags, and locations, in order to build a compound from a potentially unlimited universe of target molecules, and the mapping between tags and compounds is established only within the controlling program. This demonstrates again the universality of the programmatic synthesis approach.

V. DISCUSSION

We have formally introduced the chemtainer calculus which is capable of capturing system states and transitions of an artificial cytoplasm. The chemtainer calculus allows us to describe the organization and manipulation of chemical compounds in possibly nested, addressable bioreactors.

Elemental operations of a simple programming language are proposed, that allow for the programmatic control of chemtainer transitions. A constructive proof is given that a programming language based on eight elemental instructions is capable of constructing any possible system configuration that can be expressed in the grammar of the chemtainer calculus. This proof can serve as a core component of a compiler for automated program inference. We have presented how our machinery can be used to compile and execute complex chemical synthesis protocols and have given an example from oligosaccharide synthesis which still poses a challenging task for conventional chemical manufacturing techniques. This framework for programmable chemical synthesis demonstrates how molecular computing and chemical production can be integrated in much the same way as in biological systems, for example in the Golgi apparatus.

One might object that chemical synthesis in chemtainers works the same as sequential mixings of reaction solutions in test tubes, which is a conventional methodology. What then is the added values of chemtainers? Firstly, we point out that microfluidics in general (even without employing chemtainers), allows for the programmed set-up of reaction cascades in small volumes. This gives a general advantage over setting up reactions in test tubes either manually or even by liquid handling robots: as pointed out by Füchslin et al. [42], small spatial dimensions offer the possibility for rapid transport of intermediate compounds among different reaction environments. The accompanying reduction in processing times can be of crucial advantage in synthesis steps where intermediate compounds are only stable for short periods of time. Consequently, compounds not viable in present processing may become interesting candidates for e.g. catalysis in miniaturized systems. Employing vesicles as reaction compartments allows for the use of even smaller reaction volumes on the femtoliter scale. Secondly, DNA addressable reaction compartments can sometimes *avoid* the need of distinct reaction environments, in the sense that all reactants can be provided simultaneously in the same "pot", but individual reactions are controlled by DNA-mediated vesicle fusion. This "automated assembly" is programmed not in the microfluidic control, but in the DNA tagging of vesicle. Although this has been not exemplified in this paper, we can envision synthesis pathways where DNA tags participate in the reaction cascades, e.g., as aptamers. Folding the DNA tag set with the set of chemicals would allow chemical reactions to directly "report" about their progress, such that, e.g., a DNA tag operation is only triggered after a chemical compound has been consumed. DNA tagged reaction compartments further allow for the extraction and recovery of unreacted compounds by their unaltered DNA tag, and the extraction of reaction products by their altered DNA tag. Thirdly, encapsulation of compounds into vesicular reaction compartments offers additional advantages for chemical manufacturing in microfluidics. Compounds do not contaminate microfluidic channels as they are physically contained in vesicles. Vesicles can expose a unified physical "interface" (in terms of friction, buoyancy, charge density, etc.) for microfluidic control independent of their content. By altering the composition of membrane molecules, these properties can be altered vastly independently from the vesicle content.

In all our derivations, we have made ample use of non-deterministic semantics: we have proven that there exists a program that is able to induce desired transitions, and is thus able to construct a desired state. We have not taken into account the *likelihood* of those transitions—especially with respect to possible but undesired side reactions. As this is an important issue in the area of programming chemistry, we have carefully designed our transition system with an extension toward stochastc semantics in mind [34]. Application of these known techniques to the chemtainer calculus would be subject of future work.

ACKNOWLEDGMENTS

The research leading to these results has received funding from the Danish National Research Foundation through the Center for Fundamental Living technologies (FLinT) and from the European Community's Seventh Framework Programme (FP7/2007-2013) under grant agreements n°249032 (MATCHIT) and n°318671 (MICReagents). Steen Rasmussen, Andrew Phillips, Rasmus Petersen, Rudolf Füchslin, Ehud Shapiro, and members of the MatchIT consortium are acknowledged for helpful discussions.

[1] J. E. Rothman, Science **213**, 1212 (1981).

[2] B. Alberts, A. Johnson, J. Lewis, M. Raff, K. Roberts, and P. Watson, *Molecular Biology of the Cell* (Garland Science Publishing, 2002).

[3] J. E. Rothman, Nature **372**, 55 (1994).

[4] C. Nardin, J. Widmer, M. Winterhalter, and W. Meier, Euro. Phys. J. E **4**, 403 (2001).

[5] P.-A. Monnard, J. Membr. Biol. **191**, 87 (2003).

[6] V. Noireaux and A. Libchaber, Proc. Nat. Acad. Sci. USA **101**, 17669 (2004).

[7] R. Roodbeen and J. C. M. van Hest, BioEssays **31**, 1299 (2009).

[8] A. Richard, V. Marchi-Artzner, M.-N. Lalloz, M.-J. Brienne, F. Artzner, T. Gulik-Krzywicki, M.-A. Guedeau-Boudeville, and J.-M. Lehn, Proc. Nat. Acad. Sci. USA **101**, 15279 (2004).

[9] F. Caschera, T. Sunami, T. Matsuura, H. Suzuki, and M. Hanczyc, Langmuir **27**, 13082 (2011).

[10] H. Terasawa, K. Nishimura, H. Suzuki, T. Matsuura, and T. Yomo, Proc. Nat. Acad. Sci. USA **109**, 5942 (2012).

[11] M. Hadorn and P. E. Hotz, PLoS One **5**, e9886 (2010).

[12] M. Hadorn, E. Bönzli, P. Eggenberger Hotz, and M. M. Hanczyc, PLoS ONE **7**, e50156 (2012).

[13] M. Hadorn, E. Bönzli, H. Fellermann, P. Eggenberger Hotz, and M. M. Hanczyc, Proc. Nat. Acad. Sci. USA **109** (2012).

[14] M. Amos, P. Dittrich, J. McCaskill, and S. Rasmussen, in *Proceedings from the 2nd European Future Technologies Conference and Exhibition 2011 (FET 11)* (Procedia Computer Science, 2011) pp. 56.

[15] S. Rasmussen, A. N. Albertsen, H. Fellermann, P. L. Pedersen, C. Svaneborg, and H.-J. Ziock, in *Proceedings of the 13th Annual Conference Companion on Genetic and Evolutionary Computation (GECCO 11)* (2011) p. 15.

[16] M. Hadorn, B. Gil, C. Svaneborg, M. M. Hanczyc, H. Fellermann, R. Füchslin, P. E. Hotz, C. Kunstmann-Olsen, D. Lancet, J. McCaskill, P.-A. Monnard, G. v. Kiedrowski, and S. Rasmussen, in *Proceedings of the 13th International Conference on the Simulation and Synthesis of Artificial Life* (MIT Press, 2012).

[17] H. Fellermann, M. Hadorn, E. Bönzli, and S. Rasmussen, in *Biomimetic and Biohybrid Systems*, Lecture Notes in Computer Science No. 7375, edited by T. J. Prescott, N. F. Lepora, A. Mura, and P. F. M. J. Verschure (Springer Berlin Heidelberg, 2012) pp. 343.

[18] L. Cardelli, Nat. Comput. **10**, 407 (2011).

[19] J. S. McCaskill and P. Wagler, in *Field-Programmable Logic and Applications: The Roadmap to Reconfigurable Computing*, Lecture Notes in Computer Science No. 1896, edited by R. W. Hartenstein and H. Grünbacher (Springer Berlin Heidelberg, 2000) pp. 286.

[20] P. F. Wagler, U. Tangen, T. Maeke, and J. S. McCaskill, Biosystems **109**, 2 (2012).

[21] K. Nishimura, H. Suzuki, T. Toyota, and T. Yomo, Journal of Colloid and Interface Science **376**, 119 (2012).

[22] P. F. Wagler, U. Tangen, T. Maeke, H. P. Mathis, and J. S. McCaskill, Smart Materials and Structures **12**, 757 (2003).

[23] G. Tresset and S. Takeuchi, Biomedical Microdevices **6**, 213 (2004).

[24] Y. Tanand, K. Hettiarachchi, M. Siu, Y. Pan, and A. Lee, J. Amer. Chem. Soc. **128**, 5656 (2006).

[25] G. Paun, Journal of Computer and System Sciences **61**, 108 (2000).

[26] A. Regev, E. Panina, W. Silverman, L. Cardelli, and E. Shapiro, Theoretical Computer Science **325**, 141 (2004).
[27] L. Cardelli, in *Computational Methods in Systems Biology*, edited by V. Danos and V. Schachter (Springer, 2005) pp. 257.
[28] H. Fellermann, C. Svaneborg, S. Rasmussen, U. Tangen, T. Maeke, J. S. McCaskill, O. Markovitch, D. Lancet, D. Sorek, B. Gill, U. Shabi, E. Shapiro, R. M. Füchslin, and M. Weyland, *Integrated production/computation IT Architecture for MATCHIT*, Tech. Rep. (2013) `http://ec.europa.eu/information_society/apps/projects/logos/2/249032/080/deliverables/001_MATCHITDeliverable64Final.pdf`.
[29] R. M. Wieczorek, Le Journal de la Société des Océanistes **132**, 5 (2011).
[30] G. Stengel, R. Zahn, and F. Hk, Journal of the American Chemical Society **129**, 9584 (2007).
[31] Y. M. Chan, B. v. Lengerich, and S. G. Boxer, Proceedings of the National Academy of Sciences **106**, 979 (2009).
[32] H. Fellermann and N. Krasnogor, in *Proceedings of the Computing in Europe Conference 2014* (Budapest, 2014).
[33] H. C. Shum, J. Thiele, and S.-H. Kim, in *Advances in Transport Phenomena 2011*, Advances in Transport Phenomena No. 3, edited by L. Wang (Springer International Publishing, 2014) pp. 1.
[34] G. Bacci and M. Miculan, Theor. Comp. **431**, 117 (2012).
[35] M. Miculan and I. Sambarino, in *Proceedings of the 6th Workshop on Membrane Computing and Biologically Inspired Process Calculi MeCBIC 2012*, edited by B. Aman and G. Ciobanu (2012) p. 87.
[36] G. D. Plotkin, Journal of Logic and Algebraic Programming(2004).
[37] C. A. R. Hoare, Communications of the ACM **21**, 666 (1978).
[38] R. Milner, *A Calculus of Communicating Systems* (Springer-Verlag, New York, 1982).
[39] T. Tang, M. Y. Badal, G. Ocvirk, W. E. Lee, D. E. Bader, F. Bekkaoui, and D. J. Harrison, Anal. Chem. **74**, 725 (2002).
[40] A. Varki, Glycobiol. **3**, 97 (1993).
[41] K. M. Koeller and C. Wong, Glycobiology **10**, 1157 (2000).
[42] R. M. Füchslin, A. Dzyakanchuk, D. Flumini, H. Hauser, K. J. Hunt, R. H. Luchsinger, B. Reller, S. Scheidegger, and R. Walker, Artif. Life **19**, 9 (2013).
[43] M. S. Weyland, H. Fellermann, M. Hadorn, D. Sorek, D. Lancet, S. Rasmussen, and R. M. Füchslin, Computational and Mathematical Methods in Medicine **2013**, 467428 (2013).

Appendix A: Grammar and equivalence relations

States:

$$\mathsf{S} := \emptyset \mid \mathsf{S} \circ \mathsf{S} \mid x_i : \mathsf{P} \tag{1}$$
$$\mathsf{P} := 0 \mid \mathsf{P} + \mathsf{P} \mid \mathsf{q}^* \langle\!(\mathsf{P})\!\rangle \mid \mathsf{q} \mid m_j \tag{2}$$
$$\mathsf{q} := \mathsf{s} \mid \mathsf{s}^* \rhd \mathsf{s}^* \tag{3}$$

$$S \circ (S' \circ S'') \equiv (S \circ S') \circ S'' \tag{6}$$
$$S \circ S' \equiv S' \circ S \tag{7}$$
$$S \circ \emptyset \equiv S \tag{8}$$
$$P + (P' + P'') \equiv (P + P') + P'' \tag{9}$$
$$P + P' \equiv P' + P \tag{10}$$
$$P + 0 \equiv P \tag{11}$$
$$q_1 + (q_2 + q_3) \equiv (q_1 + q_2) + q_3 \tag{12}$$
$$q_1 + q_2 \equiv q_2 + q_1 \tag{13}$$
$$q + \diamond \equiv q \tag{14}$$
$$s_1 + (s_2 + s_3) \equiv (s_1 + s_2) + s_3 \tag{15}$$
$$s_1 + s_2 \equiv s_2 + s_1 \tag{16}$$
$$s + \diamond \equiv s \tag{17}$$
$$x_i : P \circ x_i : P' \equiv x_i : P + P' \tag{18}$$

$$\frac{S_1 \equiv S_2}{S \circ S_1 \equiv S \circ S_2} \tag{19}$$
$$\frac{P_1 \equiv P_2}{x_i : P_1 \equiv x_i : P_2} \tag{20}$$
$$\frac{P_1 \equiv P_2}{P + P_1 \equiv P + P_2} \tag{21}$$
$$\frac{q_1 \equiv q_2}{q^* + q_1 \equiv q^* + q_2} \tag{22}$$
$$\frac{s_1 \equiv s_2}{s^* + s_1 \equiv s^* + s_2} \tag{23}$$
$$\frac{s_1^* \equiv s_2^*}{s_1^* \rhd s^* \equiv s_2^* \rhd s^*} \tag{24}$$
$$\frac{s_1^* \equiv s_2^*}{s^* \rhd s_1^* \equiv s^* \rhd s_2^*} \tag{25}$$
$$\frac{P_1 \equiv P_2}{q^* \langle\!(P_1)\!\rangle \equiv q^* \langle\!(P_2)\!\rangle} \tag{26}$$
$$\frac{q_1^* \equiv q_2^*}{q_1^* \langle\!(P)\!\rangle \equiv q_2^* \langle\!(P)\!\rangle} \tag{27}$$

Transitions:

$$s_1^* \rhd s_2^* + s_1^* \longrightarrow s_2^* \tag{38}$$

$$\frac{s + s' \longrightarrow s''}{s + (s' + t) \langle\!(P)\!\rangle \longrightarrow (s'' + t) \langle\!(P)\!\rangle} \tag{39}$$

$$\frac{q + q' \longrightarrow q''}{(q + q') \langle\!(P)\!\rangle \longrightarrow q'' \langle\!(P)\!\rangle} \tag{40}$$

$$\frac{P \longrightarrow P'}{P + P'' \longrightarrow P' + P''} \tag{32}$$
$$\frac{P \longrightarrow P'}{q^* \langle\!(P)\!\rangle \longrightarrow q^* \langle\!(P')\!\rangle} \tag{33}$$
$$\frac{P \longrightarrow P'}{x_i : P \longrightarrow x_i : P'} \tag{34}$$
$$\frac{S \longrightarrow S'}{S \circ S'' \longrightarrow S' \circ S''} \tag{35}$$
$$\frac{P \equiv P' \qquad P' \longrightarrow P'' \qquad P'' \equiv P'''}{P \longrightarrow P'''} \tag{36}$$
$$\frac{S \equiv S' \qquad S' \longrightarrow S'' \qquad S'' \equiv S'''}{S \longrightarrow S'''}\,. \tag{37}$$

$$\emptyset \xrightarrow{\mathbf{feed}(x, m_i, \nu)} x : \langle\!(\nu m_i)\!\rangle \tag{41}$$
$$\emptyset \xrightarrow{\mathbf{feed\_tag}(x, s, \nu)} x : \nu s \tag{42}$$
$$x : (s + q^*) \langle\!(P)\!\rangle \xrightarrow{\mathbf{move}(s, x, z)} z : (s + q^*) \langle\!(P)\!\rangle \tag{43}$$
$$x : s \xrightarrow{\mathbf{move}(s, x, z)} z : s \tag{44}$$
$$x : s + q^* \langle\!(P)\!\rangle \xrightarrow{\mathbf{tag}(x)} x : (s + q^*) \langle\!(P)\!\rangle \tag{45}$$
$$x : q_1^* \langle\!(P)\!\rangle + q_2^* \langle\!(P')\!\rangle \xrightarrow{\mathbf{fuse}(x)} x : (q_1^* + q_2^*) \langle\!(P + P')\!\rangle \tag{46}$$
$$x : P \xrightarrow{\mathbf{flush}(x)} \emptyset \tag{47}$$
$$x : P \xrightarrow{\mathbf{wrap}(x, z)} z : \langle\!(P)\!\rangle \tag{48}$$
$$x : q^* \langle\!(P)\!\rangle \xrightarrow{\mathbf{burst}(x)} x : q^* \langle\!(\ )\!\rangle + P \tag{49}$$

Programs:

$$\pi := \epsilon \mid I; \pi \tag{51}$$
$$\epsilon ; \pi = \pi \tag{52}$$
$$(I; \pi) ; \pi' = I; (\pi ; \pi') \tag{53}$$
$$\langle \epsilon, S \rangle \longrightarrow S \tag{54}$$
$$\frac{\langle \pi, S'' \rangle \longrightarrow S \qquad I : S' \longrightarrow S''}{\langle I; \pi, S' \rangle \longrightarrow S} \tag{55}$$
$$\frac{\langle \pi, S' \rangle \longrightarrow S}{\langle I; \pi, S' \rangle \longrightarrow S} \tag{56}$$
$$\frac{S \longrightarrow S' \qquad \langle \pi, S' \rangle \longrightarrow S''}{\langle \pi, S \rangle \longrightarrow S''} \tag{57}$$
$$\frac{\langle \pi, S \rangle \longrightarrow S' \qquad S' \longrightarrow S''}{\langle \pi, S \rangle \longrightarrow S''} \tag{58}$$
$$\frac{S \equiv S'' \qquad \langle \pi, S \rangle \longrightarrow S' \qquad S' \equiv S'''}{\langle \pi, S'' \rangle \longrightarrow S'''} \tag{59}$$

Appendix B: Proofs

1. Proof of Lemma 1

We prove the statement:

$$\langle \pi, S\rangle \longrightarrow S' \quad \Longrightarrow \quad \left\langle \pi, S \circ \bar{S}\right\rangle \longrightarrow S' \circ \bar{S}. \tag{60}$$

Given that there exists an inference tree that derives the clause $\langle \pi, S\rangle \longrightarrow S'$, we show that there exists a parallel inference tree that derives $\left\langle \pi, S \circ \bar{S}\right\rangle \longrightarrow S' \circ \bar{S}$. The statement is proven by induction over the structure of these derivation.

Proof:

A. Assume that $\langle \pi, S\rangle \longrightarrow S'$ was derived by (54) in the last step. Hence, $\pi = \epsilon$ and $S' = S$. It trivially follows that $\left\langle \epsilon, S \circ \bar{S}\right\rangle \longrightarrow S \circ \bar{S}$, as axiom (54) holds for any state.

B. Assume that $\langle \pi, S\rangle \longrightarrow S'$ was derived by (55) in the last step. Hence, $\pi = I'; \pi'$.

$$\frac{\langle \pi', S''\rangle \longrightarrow S' \qquad I : S \longrightarrow S''}{\langle I; \pi', S\rangle \longrightarrow S'} \tag{55}$$

We have as induction hypothesis that $\langle \pi', S''\rangle \longrightarrow S' \Longrightarrow \left\langle \pi', S'' \circ \bar{S}\right\rangle \longrightarrow S' \circ \bar{S}$. We can then use rule (55) to derive:

$$\frac{\left\langle \pi', S'' \circ \bar{S}\right\rangle \longrightarrow S' \circ \bar{S} \qquad \dfrac{I : S \longrightarrow S''}{I : S \circ \bar{S} \longrightarrow S'' \circ \bar{S}}\,(50)}{\left\langle I; \pi', S \circ \bar{S}\right\rangle \longrightarrow S' \circ \bar{S}}\,(55)$$

Hence, if $\left\langle \pi', S'' \circ \bar{S}\right\rangle \longrightarrow S' \circ \bar{S}$ is derivable, $\left\langle I; \pi', S \circ \bar{S}\right\rangle \longrightarrow S' \circ \bar{S}$ is derivable by induction.

C. Assume that $\langle \pi, S\rangle \longrightarrow S'$ was derived by (56) in the last step. Hence, $\pi = I; \pi'$.

$$\frac{\langle \pi', S\rangle \longrightarrow S'}{\langle I; \pi', S\rangle \longrightarrow S'} \tag{56}$$

We have as induction hypothesis that $\langle \pi', S\rangle \longrightarrow S' \Longrightarrow \left\langle \pi', S \circ \bar{S}\right\rangle \longrightarrow S' \circ \bar{S}$. Rule (56) holds for any state, thus we can derive, with S substituted by $S \circ \bar{S}$:

$$\frac{\left\langle \pi', S \circ \bar{S}\right\rangle \longrightarrow S' \circ \bar{S}}{\left\langle I; \pi', S \circ \bar{S}\right\rangle \longrightarrow S' \circ \bar{S}}\,(56)$$

Hence, if $\left\langle \pi', S \circ \bar{S}\right\rangle \longrightarrow S' \circ \bar{S}$ is derivable, $\left\langle I; \pi', S \circ \bar{S}\right\rangle \longrightarrow S' \circ \bar{S}$ is derivable by induction.

D. Assume that $\langle \pi, S\rangle \longrightarrow S'$ was derived by (57) in the last step.

$$\frac{S \longrightarrow S'' \qquad \langle \pi, S''\rangle \longrightarrow S'}{\langle \pi, S\rangle \longrightarrow S'} \tag{57}$$

We have as induction hypothesis that $\langle \pi, S''\rangle \longrightarrow S' \Longrightarrow \left\langle \pi, S'' \circ \bar{S}\right\rangle \longrightarrow S' \circ \bar{S}$. We can then use rule (35) to derive:

$$\frac{\dfrac{S \longrightarrow S''}{S \circ \bar{S} \longrightarrow S'' \circ \bar{S}}\,(35) \qquad \left\langle \pi, S'' \circ \bar{S}\right\rangle \longrightarrow S' \circ \bar{S}}{\left\langle \pi, S \circ \bar{S}\right\rangle \longrightarrow S' \circ \bar{S}}\,(57)$$

Hence, if $\left\langle \pi, S'' \circ \bar{S}\right\rangle \longrightarrow S' \circ \bar{S}$ is derivable, $\left\langle \pi, S \circ \bar{S}\right\rangle \longrightarrow S' \circ \bar{S}$ is derivable by induction.

E. Assume that $\langle \pi, S\rangle \longrightarrow S'$ was derived by (58) in the last step.

$$\frac{\langle \pi, S\rangle \longrightarrow S'' \qquad S'' \longrightarrow S'}{\langle \pi, S\rangle \longrightarrow S'} \tag{58}$$

We have as induction hypothesis that $\langle \pi, S\rangle \longrightarrow S'' \Longrightarrow \langle \pi, S \circ \bar{S}\rangle \longrightarrow S'' \circ \bar{S}$. Again, we can use rule (35) to derive:

$$\frac{\dfrac{\langle \pi, S \circ \bar{S}\rangle \longrightarrow S'' \circ \bar{S} \qquad \dfrac{S'' \longrightarrow S'}{S'' \circ \bar{S} \longrightarrow S' \circ \bar{S}}\,(35)}{\langle \pi, S \circ \bar{S}\rangle \longrightarrow S' \circ \bar{S}}\,(58)$$

Hence, if $\langle \pi, S \circ \bar{S}\rangle \longrightarrow S'' \circ \bar{S}$ is derivable, $\langle \pi, S \circ \bar{S}\rangle \longrightarrow S' \circ \bar{S}$ is derivable by induction.

F. Finally, assume that $\langle \pi, S\rangle \longrightarrow S'$ was derived by (59) in the last step.

$$\frac{S \equiv S'' \qquad \langle \pi, S''\rangle \longrightarrow S''' \qquad S''' \equiv S'}{\langle \pi, S\rangle \longrightarrow S'} \tag{59}$$

We have as induction hypothesis that $\langle \pi, S''\rangle \longrightarrow S''' \Longrightarrow \langle \pi, S'' \circ \bar{S}\rangle \longrightarrow S''' \circ \bar{S}$. Thus, we can infer:

$$\frac{\dfrac{S \equiv S''}{S \circ \bar{S} \equiv S'' \circ \bar{S}}\,(19) \qquad \langle \pi, S'' \circ \bar{S}\rangle \longrightarrow S''' \circ \bar{S} \qquad \dfrac{S''' \equiv S'}{S''' \circ \bar{S} \equiv S' \circ \bar{S}}\,(19)}{\langle \pi, S \circ \bar{S}\rangle \longrightarrow S' \circ \bar{S}}\,(59)$$

Hence, if $\langle \pi, S'' \circ \bar{S}\rangle \longrightarrow S''' \circ \bar{S}$ is derivable, then $\langle \pi, S \circ \bar{S}\rangle \longrightarrow S' \circ \bar{S}$ is derivable by induction.

2. Proof of Lemma 2

We prove the statement:

$$\langle \pi', S\rangle \longrightarrow S' \quad \wedge \quad \langle \pi'', S'\rangle \longrightarrow S'' \quad \Longrightarrow \quad \langle \pi'; \pi'', S\rangle \longrightarrow S'' \tag{61}$$

Proof: Again, the proof is performed inductively over the structure of the derivation, in particular the derivation of the clause $\langle \pi', S\rangle \longrightarrow S'$ for arbitrary derivations of the clause $\langle \pi'', S'\rangle \longrightarrow S''$.

A. Assume that $\langle \pi', S\rangle \longrightarrow S'$ was derived by (54) in the last step. Hence, $\pi' = \epsilon$ and $S' = S$.

$$\langle \epsilon, S\rangle \longrightarrow S \tag{54}$$

For any π'' with $\langle \pi'', S'\rangle \longrightarrow S''$, we then have $\langle \pi'', S'\rangle \longrightarrow S'' = \langle \epsilon; \pi'', S'\rangle \longrightarrow S''$ because $\epsilon; \pi'' = \pi''$ by definition of ';' among programs.

B. Assume that $\langle \pi', S\rangle \longrightarrow S'$ was derived by (55) in the last step. Hence, $\pi' = I; \pi'''$. As induction hypothesis we have that $\langle \pi'''; \pi'', S'\rangle \longrightarrow S''$ is derivable for any π''. We can then infer that

$$\frac{\langle \pi'''; \pi'', S'\rangle \longrightarrow S'' \qquad I : S \longrightarrow S'}{\langle I; \pi'''; \pi'', S\rangle \longrightarrow S''}\,(55)$$

Hence, if $\langle \pi'''; \pi'', S'\rangle \longrightarrow S''$ is derivable, we know by induction that $\langle \pi'; \pi'', S\rangle \longrightarrow S''$ is derivable.

C. Assume that $\langle \pi', S\rangle \longrightarrow S'$ was derived by (56) in the last step. Hence, $\pi' = I; \pi'''$. We have as induction hypothesis that $\langle \pi'''; \pi'', S'\rangle \longrightarrow S''$ is derivable for any π''. We can then infer that

$$\frac{\langle \pi'''; \pi'', S\rangle \longrightarrow S''}{\langle I; \pi'''; \pi'', S\rangle \longrightarrow S''}\ (56)$$

Hence, if $\langle \pi'''; \pi'', S\rangle \longrightarrow S''$ is derivable, we know by induction that $\langle \pi'; \pi'', S\rangle \longrightarrow S''$ is derivable.

D. Assume that $\langle \pi', S\rangle \longrightarrow S'$ was derived by (57) in the last step. As induction hypothesis we have that $\langle \pi'; \pi'', S'''\rangle \longrightarrow S''$ is derivable for any π''. We can then infer that

$$\frac{S \longrightarrow S''' \qquad \langle \pi'; \pi'', S'''\rangle \longrightarrow S''}{\langle \pi'; \pi'', S\rangle \longrightarrow S''}\ (57)$$

Hence, if $\langle \pi'; \pi'', S'''\rangle \longrightarrow S''$ is derivable, we know by induction that $\langle \pi'; \pi'', S\rangle \longrightarrow S''$ is derivable.

E. Assume that $\langle \pi', S\rangle \longrightarrow S'$ has been derived by (58) in the last step:

$$\frac{\langle \pi', S\rangle \longrightarrow \bar{S} \qquad \bar{S} \longrightarrow S'}{\langle \pi', S\rangle \longrightarrow S'} \tag{58}$$

We have the second hypothesis $\langle \pi'', S'\rangle \longrightarrow S''$, and we need to show that $\langle \pi'; \pi'', S\rangle \longrightarrow S''$. By second hypothesis and (57) with $\bar{S} \longrightarrow S'$ we obtain:

$$\frac{\bar{S} \longrightarrow S' \qquad \langle \pi'', S'\rangle \longrightarrow S''}{\left\langle \pi'', \bar{S}\right\rangle \longrightarrow S''}\ (57)$$

Hence, if we have as induction hypothesis that $\langle \pi', S\rangle \longrightarrow \bar{S}$ and $\left\langle \pi'', \bar{S}\right\rangle \longrightarrow S''$ then we can follow $\langle \pi'; \pi", S\rangle \longrightarrow S''$.

F. Finally, assume that $\langle \pi', S\rangle \longrightarrow S'$ was derived by (59) in the last step:

$$\frac{S \equiv S_0 \qquad \langle \pi', S_0\rangle \longrightarrow S_1 \qquad S_1 \equiv S'}{\langle \pi', S\rangle \longrightarrow S'} \tag{59}$$

We are in the case $\langle \pi', S_0\rangle \longrightarrow S_1$ with $S \equiv S_0$ and $S_1 \equiv S'$. Our second hypothesis is that $\langle \pi'', S'\rangle \longrightarrow S''$, and we need to show that $\langle \pi'; \pi'', S\rangle \longrightarrow S''$. By (59) we then have also that $\langle \pi'', S_1\rangle \longrightarrow S''$:

$$\frac{S_1 \equiv S' \qquad \langle \pi'', S'\rangle \longrightarrow S''}{\langle \pi'', S_1\rangle \longrightarrow S''}\ (59)$$

By induction hypothesis we have that $\langle \pi', S_0\rangle \longrightarrow S_1$ and $\langle \pi'', S_1\rangle \longrightarrow S'' \Longrightarrow \langle \pi'; \pi'', S_0\rangle \longrightarrow S''$. By (59) again, $\langle \pi'; \pi'', S\rangle \longrightarrow S''$.

3. Proof of Theorem 3

Proof: by induction over the structure of S.

A. Case $S \equiv \emptyset$.
This holds in the initial state. The program to create $S \equiv \emptyset$ is ϵ: $\langle \epsilon, \emptyset \rangle \to \emptyset$.

B. Case $S \equiv S' \circ S''$ with $S', S'' \in \omega^+(\Pi_{\mathbf{min}})$.
By induction, there exist programs π' and π'' such that $\langle \pi', \emptyset \rangle \longrightarrow S'$ and $\langle \pi'', \emptyset \rangle \longrightarrow S''$. Lemma 1 then implies that $\langle \pi', \emptyset \circ \emptyset \rangle \longrightarrow S' \circ \emptyset$, and $\langle \pi'', S' \circ \emptyset \rangle \longrightarrow S' \circ S''$. It follows with lemma 2 that $\langle \pi'; \pi'', \emptyset \circ \emptyset \rangle \longrightarrow S' \circ S''$. But $\emptyset \circ \emptyset \equiv \emptyset$ and $S' \circ S'' \equiv S$, hence $\langle \pi'; \pi'', \emptyset \rangle \longrightarrow S$ due to inference rule (59).

C. Case $S \equiv x_i : P$.

C.1. Subcase $P \equiv 0$.
This holds in the initial state. The program to create $S \equiv x_i : 0$ is ϵ: $\langle \epsilon, \emptyset \rangle \to x_i : 0$.

C.2. Subcase $P \equiv P' + P''$.
This case can be reduced to case B. by means of the distributive relation (18): $x_i : P' + P'' \equiv x_i : P' \circ x_i : P''$.

C.3. Subcase $P \equiv q\langle\!(P')\!\rangle$ with $d(P') = 0$.

C.3.1. Subsubcase $P \equiv \langle\!(0)\!\rangle$.
This is achieved by the program $\pi = \mathbf{feed}(x_j, m, 0); \mathbf{transport}(x_j, x_i, x_k, \sigma)$ for arbitrary $m \in \mathcal{M}$, $x_j \in \mathcal{L}_{\mathbf{feed}}$, $x_k \in \mathcal{L}_{\mathbf{feed\_tag}}$, $\sigma \in \mathcal{T}$.

C.3.2. Subsubcase $P \equiv \langle\!(m_j)\!\rangle$.
This is achieved by the program $\pi = \mathbf{feed}(x_j, m_j, 1); \mathbf{transport}(x_j, x_i, x_k, \sigma)$ for $x_j \in \mathcal{L}_{\mathbf{feed}}$, $x_k \in \mathcal{L}_{\mathbf{feed\_tag}}$, $\sigma \in \mathcal{T}$.

C.3.3. Subsubcase $P \equiv \langle\!(s_k)\!\rangle$.
Since $d(\langle\!(s_k)\!\rangle) = 2$, we have to show that $x_i : \langle\!(s_k)\!\rangle \notin \omega^+(\Pi_{\mathbf{min}})$. Observe that **feed** and **feed_tag** either raise the nesting level of a state from 0 to 1, or leave it invariant if it was higher than 1. The instructions **tag**, **move** and **fuse** leave the nesting level unaltered, whereas **flush** reduces the nesting level to 0 or leaves it invariant. Therefore, the instruction set $I_{\mathbf{min}}$ does not contain any instruction that would increase the nesting level from 1 to 2. Therefore, $S \equiv x_i : \langle\!(s_k)\!\rangle$ is not constructable from $\emptyset$.

C.3.4. Subsubcase $P \equiv \langle\!(P' + P'')\!\rangle$.
By induction, there exists a program π such that $\langle \pi, \emptyset \rangle \longrightarrow x_i : \langle\!(P')\!\rangle \circ x_j : \langle\!(P'')\!\rangle$. Then, for some $x_k \in \mathcal{L}_{\mathbf{feed\_tag}}$, $\sigma \in \mathcal{T}$, the program $\pi; \pi'$ will produce S, where π' is the following program:

	$x_i : \langle\!(P')\!\rangle \circ x_j : \langle\!(P'')\!\rangle$
$\mathbf{feed\_tag}(x_k, \sigma, 1)$	$x_i : \langle\!(P')\!\rangle \circ x_j : \langle\!(P'')\!\rangle \circ x_k : \sigma$
$\mathbf{move}(x_k, x_j, \sigma)$	$x_i : \langle\!(P')\!\rangle \circ x_j : \langle\!(P'')\!\rangle + \sigma$
$\mathbf{tag}(x_j)$	$x_i : \langle\!(P')\!\rangle \circ x_j : \sigma\langle\!(P'')\!\rangle$
$\mathbf{move}(x_j, x_i, \sigma)$	$x_i : \langle\!(P')\!\rangle + \langle\!(P'')\!\rangle$
$\mathbf{fuse}(x_i)$	$x_i : \sigma\langle\!(P' + P'')\!\rangle$
$\mathbf{feed\_tag}(x_k, \sigma \rhd \diamond, 1)$	$x_i : \sigma\langle\!(P' + P'')\!\rangle \circ x_k : \sigma \rhd \diamond$
$\mathbf{move}(x_k, x_i, \sigma \rhd \diamond, 1)$	$x_i : \sigma\langle\!(P' + P'')\!\rangle + \sigma \rhd \diamond$
	$x_i : \langle\!(P' + P'')\!\rangle$

C.3.5. Subsubcase $P \equiv (s_k + q)\langle\!(\ P'\ )\!\rangle$.
By induction, there exists a program π such that $\langle \pi, \emptyset \rangle \longrightarrow x_i : q\langle\!(\ P'\ )\!\rangle$. Then, for any $x_k \in \mathcal{L}_{\mathbf{feed\_tag}}$, the program $\pi; \pi'$ will produce S, where π' is the program:

	$x_i : q\langle\!(\ P\ )\!\rangle$
$\mathbf{feed\_tag}(x_j, s_k, 1)$	$x_i : q\langle\!(\ P\ )\!\rangle \circ x_j : s_k$
$\mathbf{move}(x_j, x_i, s_k)$	$x_i : s_k + q\langle\!(\ P\ )\!\rangle$
$\mathbf{tag}(s_k)$	$x_i : (s_k + q)\langle\!(\ P\ )\!\rangle$

C.4. Subcase $P \equiv m_j$.
We have to show that $x_i : m_j \notin \omega^+(\Pi_{\mathbf{min}})$. First, observe that $d(x_i : m_j) = 0$. The only way to introduce m_j is by means of the command $\mathbf{feed}(x_i, m_j, 1)$ which will result in a nesting level of 1. As discussed in subcase C.3.3., the only means to decrease the nesting level is by means of the instruction **flush**. However, **flush** will transform the state $x_i : \langle\!(\ m_j\ )\!\rangle$ into the empty state, which is not equivalent to S. Thus, there is no program in $\Pi_{\mathbf{min}}$ able to generate $S \equiv x_i : m_j$. Therefore, $S \notin \omega^+(\Pi_{\mathbf{min}})$.

C.5. Subcase $P \equiv s_k$.
This is achieved by the program $\pi = \mathbf{feed\_tag}(x_j, s_k, 1); \mathbf{move}(x_j, x_i, s_k)$, for any $x_k \in \mathcal{L}_{\mathbf{feed\_tag}}$.

4. Proof of Theorem 4

Proof: The proof is identical to the one of theorem 3, where subsubcase C.3.3. is replaced by the more general subsubcase:

C.3.3' Subsubscase $P \equiv \langle\!(\ P'\ )\!\rangle$ with $d(P') > 0$, $P \not\equiv P' + P''$, and $P \not\equiv m_j$.
By induction, there exists a program $\pi \in \Pi_{\mathbf{wrap}}$, such that $\langle \pi, \emptyset \rangle \rightarrow x_j : P'$ for $x_j \in \mathcal{L}_{\mathbf{wrap\_in}}$. The desired state is obtained by the program $\pi; \pi'$, where π' is the program

	$x_j : P'$
$\mathbf{wrap}(x_j, x_k)$	$x_k : \langle\!(\ P'\ )\!\rangle$
$\mathbf{transport}(x_k, x_i, x_l, \sigma)$	$x_i : \langle\!(\ P'\ )\!\rangle$

where $x_k \in \mathcal{L}_{\mathbf{wrap\_out}}$, $x_l \in \mathcal{L}_{\mathbf{feed\_tag}}$, $\sigma \in \mathcal{T}$.

5. Proof of Theorem 5

Proof: The proof is identical to the one of theorem 4 where case C.4. is replaced by:

C.4' Case $P \equiv m_j$.
By induction, there exists a program $\pi \in \Pi_{\mathbf{burst}}$, such that $\langle \pi, \emptyset \rangle \rightarrow x_i : \sigma\langle\!(\ m_j\ )\!\rangle$. The desired state is obtained by the program $\pi; \pi'$, where π' is the program

	$x_i : \sigma\langle\!(\ m_j\ )\!\rangle$
$\mathbf{burst}(x_i)$	$x_i : \sigma\langle\!(\ )\!\rangle + m_j$
$\mathbf{move}(x_i, x_k, \sigma)$	$x_i : m_j \circ x_k : \sigma\langle\!(\ )\!\rangle$
$\mathbf{flush}(x_k)$	$x_i : m_j$

6. Proof of Theorem 6

Proof: Again, we proof the theorem by induction over the structure of S.

A. Case $S \equiv \emptyset$.
Nothing needs to be done in this case: $\langle \epsilon, \emptyset \rangle \to \emptyset$.

B. Case $S \equiv S' \circ S''$.
Just as in the proof for theorem 3, we know by induction that there exist π', π'' such that $\langle \pi', S' \rangle \to \emptyset$ and $\langle \pi'', S'' \rangle \to \emptyset$. If follows through lemmas 1 and 2 that $\langle \pi'; \pi'', S' \circ S'' \rangle \to \emptyset$.

C. Case $S \equiv x_i : P$.

C.1. Subcase $P \equiv 0$.
Nothing needs to be done in this case: $\langle \epsilon, \emptyset \rangle \to \emptyset$.

C.2. Subcase $P \equiv P' + P''$.
This is structurally equivalent to $x_i : P' \circ x_i : P''$ and is therefore reduced to case B.

C.3. Subcase $P \equiv q(P')$.

C.3.1. Subsubcase $q \equiv \diamond$.

	$x_i : (P)$
$\mathbf{feed\_tag}(x_j, \sigma)$	$x_i : (P) \circ x_j : \sigma$
$\mathbf{move}(x_j, x_i, \sigma)$	$x_i : \sigma + (P)$
$\mathbf{tag}(x_i)$	$x_i : \sigma(P)$

This reduces the problem to subsubcase C.3.2.

C.3.2. Subsubcase $q \equiv s_k + q'$.
For $x_j \in \mathcal{L}_{\mathbf{flush}}$, the following program resets the state:

	$x_i : (s_k + q)(P)$
$\mathbf{move}(x_i, x_j, s_k)$	$x_j : (s_k + q)(P)$
$\mathbf{flush}(x_j)$	$\emptyset$

C.4. Subcase $P \equiv m_j$.
If $x_i \in \mathcal{L}_{\mathbf{flush}}$, the program $\mathbf{flush}(x_i)$ will reset the state. Likewise, if $x_i \in \mathcal{L}_{\mathbf{wrap\_in}}$, the program $\mathbf{wrap}(x_i, x_j)$ will transform the state into $x_j : (m_j)$ for any $x_j \in \mathcal{L}_{\mathbf{wrap\_out}}$ and reduces the problem to subcase C.3.1.. On the other hand, if $x_i \notin \mathcal{L}_{\mathbf{flush}} \cup \mathcal{L}_{\mathbf{wrap\_in}}$, we have to show that S is not an element of $\omega^-(\Pi_x)$. Note that all instructions **feed**, **feed_tag tag**, **move**, **burst** leave $x_i : m_j$ invariant under transition. Thus, there is no sequence of instructions that would transform $x_i : m_j$ into the empty state. Therefore, $x_i : m_j \notin \omega^-(\Pi_x)$ for $x_i \notin \mathcal{L}_{\mathbf{flush}} \cup \mathcal{L}_{\mathbf{wrap\_in}}$.

C.5. Subcase $P \equiv s_k$.
For $x_j \in \mathcal{L}_{\mathbf{flush}}$, the following program resets the state:

	$x_i : s_k$
$\mathbf{move}(x_i, x_j, s_k)$	$x_j : s_k$
$\mathbf{flush}(x_j)$	$\emptyset$