A Chemical Compiler for the Synthesis of Branched Oligomers on Standardized Chemical Reaction Structures

Mathias S. Weyland[1], Dandolo Flumini[1], Johannes J. Schneider[1],
Alessia Faggian[2], Aitor Patiño Diaz[2] and Rudolf M. Füchslin[1,3]

[1]Institute of Applied Mathematics and Physics, School of Engineering, 8401 Winterthur, Switzerland
[2]Laboratory for Artificial Biology, Dept. of Cellular, Computational and Integrative Biology, University of Trento, 38123 Povo, Italy
[3]European Centre for Living Technology, 30124 Venice, Italy
mathias.weyland@zhaw.ch

Abstract

This research paper presents a chemical compiler developed to find optimal configurations of a platform for synthesizing specific branched oligomers in an artificial chemistry, along with exemplary compiler output and benchmarks where the platform configuration suggested by the compiler is compared to other configurations in simulation. The compiler operates as a pipeline with two stages: labelling and optimization. The report explains the structure of the compiler target and its interpretation, followed by a code walk-through of the compiler stages with code snippets and examples. The compiler can be used as a code generator for reactions in a chemical simulator and to derive loading schemes for multi-level droplets. The results obtained in simulations suggest that the container system can efficiently optimize the yield of coupled reaction networks and that multi-level droplets can lead to significant improvements.

Introduction

In this paper, we adopt the term "chemical compiler" to refer to a computer program that generates specifications of initial conditions for a theoretical or actual chemical reaction platform. Given a desired behaviour of the chemical system as input, the chemical compiler generates instructions on how to set up the system so that the behaviour can be observed in the subsequent reaction dynamics. In this work specifically, the reaction platform targeted by the compiler is a theoretical framework consisting of a linear chain of reaction chambers. Given a description of a branched oligomer as input, the chemical compiler generates a specification of a sequence of reaction chambers, each pre-filled with suitable reactants, to form a chemical reaction platform capable of synthesizing the desired molecule with a high yield rate. This is particularly interesting in a setting where byproducts would compromise the yield. Our compiler prevents the formation of undesired byproducts by suitably arranging the reaction chambers.

While this work is deliberately based on a reaction platform that remains abstract, we suggest that, e.g., a cascade of microfluidic reactors (cf. Utada et al. (2007)) would be a suitable implementation of the platform. Alternatively, the reactands could be encapsulated and compartmentalized within liposomes with the ability of controlled merging (cf. Angelova and Dimitrov (1986)). Both approaches are promising candidates as they would enable confined reactions in microreactor-like environments. In particular, we would like to implement the compiler output using the droplet structures developed in the ACDC project (cf. Li et al. (2022)) in the future.

The high-level workflow of the compiler is as follows:

1. The users specify a desired molecule (in our case, a polymer) they wish to synthesize.
2. The chemical compiler generates a suitable setup specification that will be able to synthesize the target molecule with a high yield rate.
3. The users set up the physical system according to the chemical compiler's specifications and "run" the system to produce the desired molecule.

The compiler presented here is inspired by graph-theoretical approaches to chemical reaction networks (cf. Temkin et al. (1996) for an overview) and is compatible with a framework outlined in Weyland et al. (2020) which aims at deriving chemical reaction platforms with maximum yield. It extends the compiler by Weyland et al. (2013) in that it aims at reducing the number of reaction chambers needed.

In the subsequent sections, this paper provides a detailed description of the chemical compiler and its capabilities. Specifically, in the following section, we define the compilation target, i.e., the chemical reaction system, and discuss the underlying assumptions and constraints of the system. In the section about the compiler pipeline, we present the details of how the chemical compiler generates the chemical reaction system from a description of a branched oligomer. This section includes a step-by-step walk-through of the compiler stages: labelling and optimization, along with code snippets and examples. To evaluate the effectiveness of the chemical compiler, in the last section, we present the results of several simulations that compare the performance of the compiler's output to manual setups in terms of synthesis time and yield

rate. We demonstrate that the chemical compiler can efficiently optimize the yield of coupled reaction networks.

The Compiler Target and Its Intended Dynamics

Compiler Target

As mentioned in the introduction, the target of our chemical compiler consists of specifications for specific chemical reaction networks. This section defines "standardized chemical reaction structures (SCR structures)" that our compiler generates as output.

A (labelled) monomer is a pair (a, b) consisting of a monomer label "a" and a (finite) list of bonds "b". An SCR structure is a (finite) list of sets of monomers. We denote SCR structures as tuples of the form

$$(\{a_1[b_1^1, \ldots b_1^{n_1}], \ldots, a_k[b_k^1, \ldots b_k^{n_k}]\}, \ldots)$$

where $a_i[b_i^1, \ldots, b_i^{n_i}]$ denotes a monomer with label a_i and bonds $b_i^1, \ldots, b_i^{n_i}$.

In SCR structures, labelled monomers represent the basic reactants of a chemical reaction system. Each labelled monomer has specific binding sites that are available for reactions. Connecting two labelled monomers at a specific binding site requires the presence of a free binding site with an identical label on each monomer and then combining these binding sites into a bond. Binding sites that pertain to a bond are no longer free. Structures obtained from connecting two or more labelled monomers are partial molecules if free binding sites are left or labelled molecules otherwise.[1] Finally, an SCR structure represents a linear arrangement of reaction compartments, such as vesicles, droplets, or containers. Each compartment is preloaded with a set of labelled monomers. The order in which the SCR structure arranges the compartments specifies how the labelled monomers can flow between the chambers and interact (bind to each other).

In software, we use the following Haskell datatypes to represent SCR structures:

```
data LabeledMonomer = LMonomer Monomer [Bond]
data SCRStruct = SCRStruct [Set LabeledMonomer]
```

where `Monomer` and `Bond` are aliases for strings.

Intended Interpretation and Dynamics

In this paragraph, we describe the dynamics of SCR structures as they go through various configurations when monomers move between reaction chambers and bind together.

We assume that the pores that connect compartments in a linear SCR structure can be opened by an external signal, which opens the pores one by one, starting from the first to the last. This signal can be thought of as a physical or chemical stimulus, such as light or a specific chemical compound. As the synthesis progresses and more monomers bind together, increasingly complex structures form until the target molecule is synthesized with a high yield rate (cf. Fig. 1). The chemical compiler's task is to derive a suitable initial structure that will improve the yield rate of the target molecule.

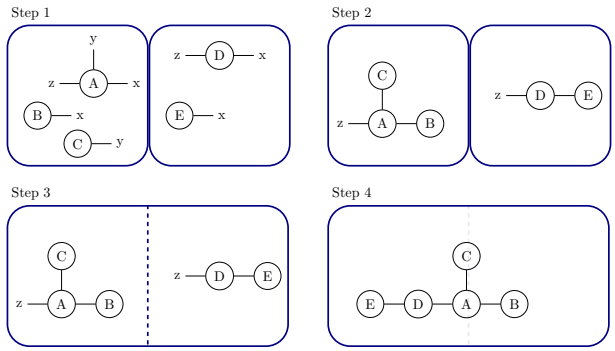

Figure 1: Four steps in the evolution of an initial SCR structure. Step 1: Initialization; step 2: inner cell reactions forming intermediate products; step 3: Formation of pores; step 4: reactions forming complete molecules.

In the absence of an external signal to open the pores, we adopt a nondeterministic interpretation of the dynamics, where the pores are always open, and the intermediate products of the reaction may flow freely between the connected compartments. While this interpretation does not guarantee an optimal outcome, the yields are high due to the proximity of preferred reaction pairs. In simulations, we applied this nondeterministic interpretation and obtained almost optimal yield rates (see section "Simulations").

As an example of how different initial structures may provide different yield rates, consider the example shown in Fig. 1. The displayed initial setup provides better yield rates than a more straightforward setup where all monomers are in a single compartment because it reduces the formation of undesired reaction byproducts such as for example "E-x-B" which can reduce overall yield rates.

The Compiler Pipeline

This section provides an overview of the compiler's pipeline, which begins with a description of a molecule (polymer) and proceeds to generate a labelled graph of its connections (labelling stage). The compiler then uses the generated labelled graph and generates an SCR structure that can optimistically synthesize the target molecule in the optimization stage. Before delving into the compiler's processing stages, we briefly define the compiler's input. The compiler's input is a molecule (or polymer), represented as a graph comprising nodes and edges. Each node represents a monomer, and

[1] While we do not impose a particular chemical implementation of this scheme, click chemistries such as the ones described by Kolb et al. (2001) may satisfy the requirements stated here.

each edge represents a bond between monomers. The labels on nodes and edges provide information about the types of monomers and bonds present in the molecule. However, the input for the compiler requires that the edges are not labelled. Below is a simplified[2] representation of the datatypes we use to represent molecules in code, where we model a graph in terms of a mapping associating edges to unordered pairs of nodes.

```
data Molecule = Molecule
  { mBonds :: Map bondId bond
  , mMonomers :: Map monomerId monomer
  , mStructure :: Map (UnorderedPair monomerId) bondId
  }
```

Finally, we use `()` as bond type for input molecules, which are molecules that lack edge labels.

The Labelling Stage

The labelling stage of the compiler begins with an unlabeled molecule and a set of given linkers or bonds. It distributes the linkers over the connections in the molecule such that no monomer is adjacent to two connections with the same label. This distribution is a crucial first optimization step, as it ensures that the desired molecule is synthesized correctly. Consider the following example to understand why such a distribution is necessary. Suppose we want to synthesize the molecule shown in Fig. 3 and label all connections adjacent to "A" with the same label 'x'. Then, we may inadvertently synthesize an artefact like the one shown in Fig. 2 instead of our target molecule. It is important to note that which of the potential faulty molecules we end up synthesizing and in what ratios depends on the order of the synthesis steps.

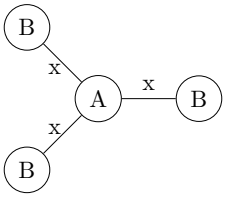

Figure 2: One of the possible undesired molecules that occur instead of the molecule shown in Fig. 3 when the "A"-monomer's bonds are all labelled with identical labels, e.g. 'x'.

The labelling algorithm takes as input a desired molecule and a set of available linkers and processes these inputs as follows:

1. Initially, associate each monomer in the molecule with the set of all available linkers.
2. For every connection in the molecule, perform the following steps:
   (a) Find the intersection of the linker sets associated with the endpoints of the connection. If the intersection is empty, the algorithm fails. Otherwise,
   (b) Choose a linker from the intersection found in step (a) and associate it with the connection.
   (c) Remove the chosen linker from the sets associated with the endpoints of the connection.

Note that the runtime of this algorithm is linear in the number of connections in the molecule. Our implementation of the algorithm in Haskell consists of a main function with the signature:

```
labelMSkeleton :: MSkeleton -> Set Bond -> Molecule
```

Here, `MSkeleton` and `Molecule` respectively refer to input molecules (i.e., molecules without edge labels, i.e. "molecule skeletons") and labelled molecules. See Figs. 3 and 4 for an example of an input and the respective output generated by the labelling function with linker set $\{x, y, z\}$.

The Optimization Stage

The optimization stage aims to generate an initial SCR structure from a labelled molecule. The compiler optimizes the yield rate of the desired molecule by generating a structure that ensures that undesired reaction pairs are either never located in the same cell or are less likely to be so (non-deterministic case). As a secondary objective, the compiler also attempts to minimize the length of the resulting SCR structure. The optimization process begins by decomposing a labelled molecule into labelled monomers, where each monomer is associated with its respective linkers as they appear in the labelled molecule. Next, the compiler distributes these monomers into compartments, making sure not to place monomers not connected in the target molecule but with identical linkers in the same compartment. At the core of this functionality is the following simple function:

```
partition :: (a -> a -> Bool ) -> [a] -> [[a]]
partition f xs = case xs of
   [] -> []
   x:xs -> insert_ x $ partition f xs
   where
      insert_ x [] = [[x]]
      insert_ x (ys:yss)
         | all (f x) ys = (x:ys):yss
         | otherwise = ys:insert_ x yss
```

The partition function takes a binary predicate and a list of elements as input. It then creates a list of lists, where each sublist contains elements that are pairwise compatible according to the given predicate. The function achieves this by iterating through the input list and placing each element in the first sublist where it is compatible with all existing elements. If no such sublist exists, it creates a new sublist.

[2]Throughout this paper, we use notations similar to Haskell code. However, we may omit some details, such as type parameters, and occasionally rename elements to improve readability and fit within the two-column layout.

The algorithm engages in a worst-case scenario of 'k' comparisons for each element 'x' at position 'k' in the list before successfully inserting 'x' into the target structure. Consequently, the cumulative runtime of the algorithm scales quadratically with the total number of elements in the input list. The list of sublists generated is assured of achieving minimal length in cases where the given predicate is an equivalence relation. Although minimum length is not assured when utilizing other predicates, the algorithm often yields an optimal solution regardless. The remaining code of the optimization stage of the compiler focuses on generating and connecting appropriate datatypes and specifying suitable parameters to implement the two primary functions:

```
compartmentalize :: Molecule -> SCRStruct

selectiveCompartmentalize
   :: (Monomer -> Monomer -> Bool)
   -> Molecule
   -> SCRStruct
```

The `compartmentalize` function utilizes the `partition` function with a predicate `compatibleIds` to ensure that monomers with a shared bond can only be combined in a compartment if they are connected in the target molecule. The implementation of the predicate is as follows:

```
compatibleIds
  :: Molecule
  -> [BondId]
  -> [BondId]
  -> Bool
compatibleIds m bis1 bis2 =
  and [ x == y || g x y | x <- bis1, y <- bis2]
  where
    g bi1 bi2 = lookup bi1 (mBonds m)
              /= lookup bi2 (mBonds m)
```

The `selectiveCompartmentalize` function is a more general version of `compartmentalize` that enables users to introduce additional constraints, such as preventing certain bonds from forming in the same location as other bonds. Finally, Running the complete pipeline simply means combining the compiler stages in one function:

```
generateStructure
   :: MSkeleton
   -> Set Bond
   -> SCRStruct
generateStructure m bonds = compartmentalize
  $ labelMSkeleton m bonds
```

Simulations

In this section, we provide example input and output for all stages of the compiler pipeline. Furthermore, we showcase the full compiler pipeline in an example and compare the resulting synthesis platform configuration to "hand-crafted" alternatives.

Applying the compiler pipeline

To showcase our chemical compiler, we use it to produce the SCR structure for synthesising a target polymer from five (abstract) monomers "A" to "E", cf. Fig. 3.

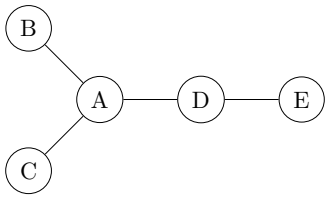

Figure 3: The target polymer consisting of five monomers "A" to "E".

In code, this target polymer consists of five unlabelled bonds with `bondId` 0 to 4, "A" to "E" with `monomerId` 0 to 4 and the mapping of the bonds to the monomers according to Fig. 3. It is specified as follows:

```
targetMolecule = Molecule
  { mBonds = fromList
    [ ("0",()), ("1",()), ("2",()), ("3",()), ("4",()) ]
  , mMonomers = fromList
    [ ("0","A")
    , ("1","B")
    , ("2","C")
    , ("3","D")
    , ("4","E")
    ]
  , mStructure = fromList
    [ (Uop "2" "0", "0") -- C–A
    , (Uop "1" "0", "1") -- B–A
    , (Uop "0" "3", "2") -- A–D
    , (Uop "3" "4", "3") -- D–E
    ]
}
```

When provided with this input, the compiler's labelling stage produces a labelled target molecule with the labels 'x', 'y' and 'z' that is efficiently synthesizable (cf. Fig. 4. In particular, the compiler refrained from re-using any labels when labelling the bonds of "A", as these would inevitably yield undesired by-products. Conversely, the 'x' label was re-used since the creation of "ExB" artefacts can be avoided as described previously.

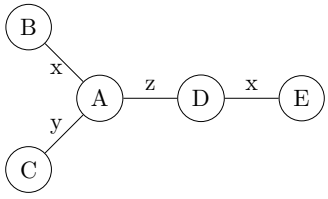

Figure 4: Output of the labelling stage: The target polymer with labelled bonds.

Now, we consider two different scenarios, each with a different set of constraints regarding intermediate products: (1) In the first scenario, there are no restrictions, i.e. all intermediate products are admissible. (2) In the second scenario, the intermediate product "AxB" is considered toxic and shall

never occur in the synthesis path. Thus, the compiler is required to generate an SCR structure that guarantees the absence of the toxic intermediate product during the synthesis of the target polymer.

In scenario (1), the compiler generates a structure with two compartments, one loaded with monomers “A” to “C” and another with monomers “D” and “E”.

```
({A[x,y,z],B[x],C[y]},{D[z,x],E[x]})
```

As stated previously, a minimum of two compartments is required to prevent the creation of “ExB” byproducts. The synthesis on this SCR structure runs as shown in Fig. 1.

In scenario (2), we provide the compiler with information about the supposed “incompatibility relation” of monomers “A” and “B” and formalize this as the function

```
\m1 m2 -> Uop m1 m2 /= Uop "A" "B"
```

which we pass to the compiler. Consequently, the compiler generates the following structure with three compartments where “B” is initialized in a dedicated compartment to satisfy the additional constraints.

```
({A[x,y,z],C[y]},{B[x]},{D[z,y],E[x]})
```

Dynamic simulation and comparison with other platform configurations

To showcase the leverage of our compiler, we use a simulator described by Schneider et al. (2020, 2021, 2022) which simulates the assembly of a network of droplets and the subsequent reactions of the chemicals loaded into the droplets. In brief, 3000 spherical droplets with a radius of 30 µm are initialized within a cylinder of radius 1 mm and height of 4 mm so that they do not touch or intersect with the cylinder walls or with each other. The droplets are subject to gravitational, frictional and buoyancy forces and, in consequence, agglomerate at the bottom of the cylinder. Two droplets that touch can form pores, thus exchanging chemicals which react according to the chemistry derived by the simulator (we assume no immediate toxic products in this example; the simulation of the reactions is governed by algorithm developed by Gillespie (1976, 1977)).

We investigate a one-pot-reaction baseline (pot) and five different scenarios defined as follows: (s1) Five types of droplets are loaded with one monomer (“A” to “E”) each, and pores are formed wherever droplets touch, regardless of their type. (s2) The second scenario is the same as the first scenario, with the exception that droplets of the same type do not form pores. This scenario is expected to perform similarly to the first one. (s3) The third scenario again uses five types of droplets loaded with the respective monomers as before, but pores only form between droplets with the following (original) content: “A” and “C”, “B” and “C”, “C” and “D” and “D” and “E”. The third scenario is crafted in an attempt to guide the gradual reaction process according to the structure of the target polymer. (s4) The fourth scenario is similar to the third scenario but pores also form between droplets of the same type. (s5) In the fifth scenario, pores always form except for “C” and “E” as well as “A” and “D”. This is crafted in an attempt to avoid undesired byproducts.

The aforementioned scenarios are compared to two “improved” scenarios. (imps1) allows for unconditional pore formation between *any* type of droplets, while (imps2) allows for pore formation between *different* types of droplets. In contrast to the previous scenarios, the improved scenarios employ droplets that are loaded according to the SCR structures proposed by the chemical compiler. Thus, only two types of droplets are used.

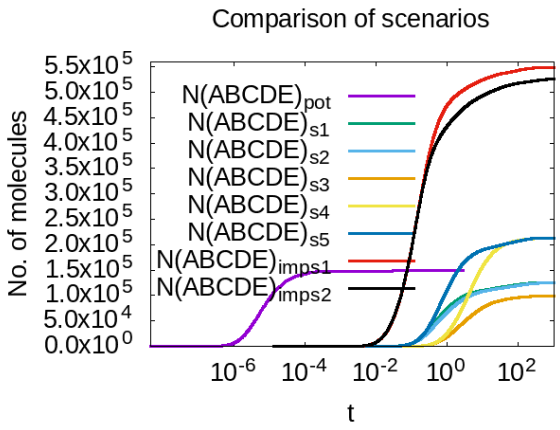

Figure 5: Simulation results for the one-pot reaction (pot), scenarios (s1) to (s5) and the scenarios improved by the compiler, (imps1) and (imps2). The latter produce a much higher yield compared to any other scenario, and, with the exception of the one-pot reaction, are also faster.

The number of target polymers for any of these scenarios is shown in Fig. 5 as a function of simulation time. The one-pot reaction is very fast since any reaction can occur immediately, in contrast to all other scenarios where a droplet agglomeration with pore formation is a prerequisite for the synthesis of the target polymer. Furthermore, the yield is higher than in scenarios (s1) to (s3). Scenarios (s4) and (s5) produce similar yields, but the dynamics in (s4) are faster than in (s5). The results from the simulations of the improved scenarios are better than the others in two aspects. First, the dynamics are faster, except for the one-pot baseline. Second, the yield of all other scenarios is dwarfed by the yield of the two improved scenarios that were suggested by the compiler presented in this work.

To conclude, we gave an outline of the chemical compiler and demonstrated how it could be used to achieve high yields in a short amount of time. We see the potential for this approach in situations where reactants are unstable and

may degrade quickly in reaction systems that are prone to the formation of undesired byproducts or toxic intermediate products.

Acknowledgements

The compiler presented in this work was developed within the ACDC project, which has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 824060.

References

Angelova, M. I. and Dimitrov, D. S. (1986). Liposome electroformation. *Faraday discussions of the Chemical Society*, 81:303–311.

Gillespie, D. T. (1976). A general method for numerically simulating the stochastic time evolution of coupled chemical reactions. *Journal of computational physics*, 22(4):403–434.

Gillespie, D. T. (1977). Exact stochastic simulation of coupled chemical reactions. *The journal of physical chemistry*, 81(25):2340–2361.

Kolb, H. C., Finn, M., and Sharpless, K. B. (2001). Click chemistry: diverse chemical function from a few good reactions. *Angewandte Chemie International Edition*, 40(11):2004–2021.

Li, J., Jamieson, W. D., Dimitriou, P., Xu, W., Rohde, P., Martinac, B., Baker, M., Drinkwater, B. W., Castell, O. K., and Barrow, D. A. (2022). Building programmable multicompartment artificial cells incorporating remotely activated protein channels using microfluidics and acoustic levitation. *Nature Communications*, 13(1):4125.

Schneider, J. J., Faggian, A., Holler, S., Casiraghi, F., Li, J., Cebolla Sanahuja, L., Matuttis, H.-G., Hanczyc, M. M., Barrow, D. A., Weyland, M. S., Flumini, D., Eggenberger Hotz, P., and Füchslin, R. M. (2021). Influence of the geometry on the agglomeration of a polydisperse binary system of spherical particles. In *Artificial Life Conference Proceedings*, volume 33. MIT Press. Alife 2021 : The 2021 Conference on Artificial Life, online, 19-23 July 2021.

Schneider, J. J., Weyland, M. S., Flumini, D., and Füchslin, R. M. (2022). Exploring the three-dimensional arrangement of droplets. In Schneider, J. J., Weyland, M. S., Flumini, D., and Füchslin, R. M., editors, *Artificial Life and Evolutionary Computation*, pages 63–71, Cham. Springer Nature Switzerland.

Schneider, J. J., Weyland, M. S., Flumini, D., Matuttis, H.-G., Morgenstern, I., and Füchslin, R. M. (2020). Studying and simulating the three-dimensional arrangement of droplets. In Cicirelli, F., Guerrieri, A., Pizzuti, C., Socievole, A., Spezzano, G., and Vinci, A., editors, *Artificial Life and Evolutionary Computation*, pages 158–170, Cham. Springer International Publishing.

Temkin, O. N., Zeigarnik, A. V., and Bonchev, D. (1996). *Chemical reaction networks: a graph-theoretical approach*. CRC Press.

Utada, A., Chu, L.-Y., Fernandez-Nieves, A., Link, D., Holtze, C., and Weitz, D. (2007). Dripping, jetting, drops, and wetting: The magic of microfluidics. *Mrs Bulletin*, 32(9):702–708.

Weyland, M. S., Fellermann, H., Hadorn, M., Sorek, D., Lancet, D., Rasmussen, S., and Füchslin, R. M. (2013). The matchit automaton: exploiting compartmentalization for the synthesis of branched polymers. *Computational and mathematical methods in medicine*, 2013.

Weyland, M. S., Flumini, D., Schneider, J. J., and Füchslin, R. M. (2020). A compiler framework to derive microfluidic platforms for manufacturing hierarchical, compartmentalized structures that maximize yield of chemical reactions. In *Proceedings of the Artificial Life Conference 2020*, pages 602–604. Massachusetts Institute of Technology. International Conference on Artificial Life (ALIFE), online, 13-18 July 2020.