TECHNISCHE UNIVERSITEIT DELFT

MASTER OF SCIENCE THESIS IN COMPUTER SCIENCE

Automated Discovery of Chemical Reaction Networks using Program Synthesis

Author:
Richard WIJERS

Supervisors:
Dr. Sebastijan DUMANČI
Ir. Reuben GARDOS REID
Ć

9th July 2025

Delft University of Technology

Automated Discovery of Chemical Reaction Networks using Program Synthesis

Master's Thesis in Computer Science

Algorithmics group
Faculty of Electrical Engineering, Mathematics, and Computer Science
Delft University of Technology

Richard Wijers

9th July 2025

Author
Richard Wijers

Title
Automated Discovery of Chemical Reaction Networks using Program Synthesis

MSc presentation
16th July 2025

Graduation Committee

Dr. Jasmijn Baaijens	Delft University of Technology
Dr. Neil Yorke-Smith	Delft University of Technology
Ir. Reuben Gardos Reid	Delft University of Technology

Abstract

This thesis explores the automated construction of Chemical Reaction Networks (CRNs) from incomplete experimental data, a task traditionally dependent on expert knowledge and manual effort. CRNs model the interactions between chemical species through a network of reactions and are essential in fields such as medicine and chemistry. However, many real-world systems include unobserved or unmeasurable species, making CRN construction challenging. To address this, this thesis frames CRN discovery as a program synthesis problem, using grammars and constraints to define the space of possible CRNs. A modular synthesis pipeline is developed that incrementally builds candidate molecules, reactions, and networks given a problem definition. Experimental results demonstrate that constraints effectively reduce the search space and that the solver is capable of identifying the correct reaction networks. Moreover, a scoring mechanism ranks the expected CRN highly among generated candidates.

Contents

Chapter 1

Introduction

Many advancements in medicine and chemistry rely on accurate models of chemical systems to ensure that they behave in a predictable and stable way. For example, in medicine development a requirement could be that the working substance should not react with other medicine or herbs taken by the patient [Zhou et al., 2021]. By improving our understanding of how all the substances react together, more precise adjustments can be made with better results. [Wen et al., 2023]

Since in reality systems, for example of the just discussed medicine, can become very complex, they are modelled in a simplified system. One type of model used to represent these systems is a chemical reaction network (CRN), which contains a list of reactions describing how the substances (called species) interact with each other. However, building such models is a difficult task, especially when some parts of the system are not directly observed or measured.

For example, when a scientist is performing experiments to understand more about underlying CRN, it might not be possible to measure every species that is active in the network. Completing the missing information in those networks is often labour intensive and relies mainly on expert insights. However, recent advances in technology allow for the opportunity to research chemistry beyond the limitations of manual exploration [Unsleber and Reiher, 2020]. This thesis explores using program synthesis techniques to automate this construction of chemical reaction networks.

To illustrate this idea, consider the following simple reaction for creating water: $2\,H_2 + O_2 \longrightarrow 2\,H_2O$. Given that we are only able to measure O_2 and H_2O, can we without knowing the reaction, deduce that H_2 was also present in the network and what its concentration over time looks like? This problem is depicted in figure 1.1, where the measured molecules are the bold lines, and the concentration line, which we have to find, is displayed with a dashed line.

In this example we have a problem with two known molecules, where O_2 is decreasing and H_2O is increasing, with their corresponding concentration profiles. And we have to find the CRN that describes this behaviour as closely as possible.

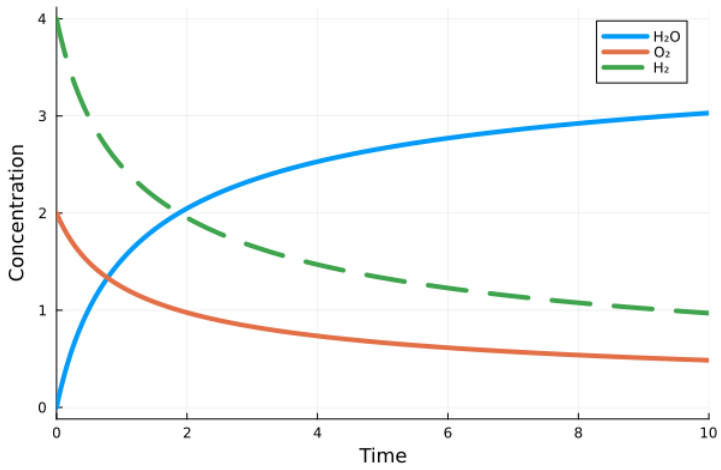


Figure 1.1: Example concentration graph for water synthesis. The red and orange lines represent the known species water and oxygen. The green dashed line indicates the unknown specie, hydrogen.

Since this is a small example, we can easily fill in the missing information $? + O_2 \longrightarrow 2\,H_2O$ with $2\,H_2$ to complete a reaction. However, when the molecules become more complex and there are multiple and bigger reactions in the network, there are many options to consider, and manually finding and evaluating these networks becomes impractical.

To address this, we can turn to automated approaches where this big search space is explored. One of these approaches is program synthesis, where using a set of input-output examples, a program is synthesized that satisfies the example. [David and Kroening, 2017]

By framing the chemical reaction as a ”program”, we can leverage this program synthesis method. This thesis investigates how such techniques can be applied to automate the construction of chemical reaction networks, particularly in cases where data is incomplete.

This will be done by answering the following questions:

- Do constraints successfully narrow down the search space?
- Is the solver able to find the correct chemical reaction network?
- Does the expected solution receive the highest score?

Chapter 2

Background

This chapter covers the two main concepts that this thesis makes use of: Chemical Reaction Networks (CRNs) and Program Synthesis. Understanding these concepts is important for following the rest of the work as they form the basis of the methods and contributions discussed later on. The goal here is to give enough background so that the thesis can be fully understood without separate knowledge. This chapter will start with an introduction to CRNs; what they are, how they are represented, and how they can be simulated. After which, program synthesis will be explained, including grammars, search strategies, and constraints.

2.1 Chemical Reaction Networks

Chemical Reaction Networks (CRNs) provide a formal way to describe how different substances, called species, interact through reactions. This section first introduces the basic components of a CRN and its reactions. Since in this thesis species mainly refer to molecules, we will also briefly explain the SMILES notation used to describe these molecules. Finally, an outline of what a simulation of a CRN looks like will be discussed.

2.1.1 Basic structure

A chemical reaction network consists of:

- **Species**: The different entities (molecules) involved in the reactions, such as oxygen or water molecules.
- **Reactions**: Processes that describe the change of one set of species into another, for example, oxygen and hydrogen combining to create water.

Each reaction can be represented in the general form:

$$\mathrm{r}, \alpha_1 \, \mathrm{X}_1 + \alpha_2 \, \mathrm{X}_2 + ... \longrightarrow \beta_1 \, \mathrm{Y}_1 + \beta_2 \, \mathrm{Y}_2 + ...$$

where:

- r is the rate at which this reaction occurs.
- $X_1, X_2, \ldots$ are the reactant species (inputs) and $Y_1, Y_2, \ldots$ are the product species (outputs).
- $\alpha_1, \alpha_2, \ldots$ and $\beta_1, \beta_2, \ldots$ are the coefficients indicating the number of each species involved.

CRNs are widely used in fields such as chemistry and biology to describe complex systems. By analysing the structure of a CRN, researchers can gain insights into the underlying mechanisms.

2.1.2 SMILES Notation

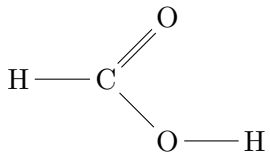
Two molecules with the same number of atoms can have a different effect based on the structure. Therefore, a notation is needed that can reflect this structure. In this thesis the SMILES (Simplified Molecular Input Line Entry System) notation is chosen, a widely used string-based format for describing molecular structures. For instance, the water molecule H_2O can be represented in SMILES as `[H]-[O]-[H]`. There are many ways to use implicit notation in SMILES, however this thesis focusses on a smaller subset of explicit rules.

The following rules are used:

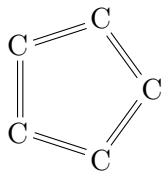
- Atoms are written in square brackets, e.g., `[C]`, `[O]`.
- Bonds between atoms are denoted using characters like - (single), = (double).
- Branches are denoted with parentheses, allowing more than two connections to a single atom. For instance NH_3 can be written as `[H]-[N](-[H])-[H]`
- Rings are encoded using matching digits to indicate connection points. If there are multiple rings in a molecule, different digits can be used for each ring connection.

Each atom can form a specific number of bonds. For instance, a carbon atom (C) typically forms four bonds, oxygen (O) forms two, nitrogen (N) forms three, and hydrogen (H) forms only one. These bonding rules must be respected when constructing valid molecular structures. For example, a carbon atom written as `[C]` must be connected to a total of four other atoms or bonds, such as in methane `[H]-C(-[H])(-[H])-[H]`.

To visualize how the SMILES notation maps to a molecular structure, consider the molecule `[H]-[C](=[O])-[O]-[H]`, which corresponds to:



For a small ring example, the SMILES string `[C]=1=[C]=[C]=[C]=[C]=1` is a ring of carbon molecules where the first and last atom are connected with the digit 1. results in the following structure:



The SMILES notation supports multiple different properties that are not used in this thesis. For more information you can check the OpenSMILES specification [1]

2.1.3 Simulations

A simulation of a CRN gives an overview of how the concentrations of molecules change over time. Take for instance the small system for synthesizing water: $0.1, 2\,H_2 + O_2 \longrightarrow 2\,H_2O$. This system describes that every unit of time, 10 percent of the molecules will undergo this reaction of hydrogen and oxygen reacting into water. The reaction will be limited by the lower concentration, meaning that if there is an infinite amount of hydrogen, but only 100 molecules of oxygen, the reaction will only occur 10 times at the first time slot. Note that for a single reaction, two hydrogen molecules are needed, thus to get the same number of reactions in the first time slot 200 molecules of hydrogen are needed. An example of how the concentration would look over time is in figure 2.1

As discussed before, not always will everything be completely known. To solve this, parameter estimation can be used. Given a CRN without the reaction rates and initial concentrations, we can use a solver to tune these to match a measurement. To illustrate this example, just a few points are measured and we can find the closest matching line as seen in figure 2.2. These simulations are done using Catalyst.jl [2].

[1] OpenSMILES specification: http://opensmiles.org/

[2] Catalyst.jl: https://docs.sciml.ai/Catalyst/stable/

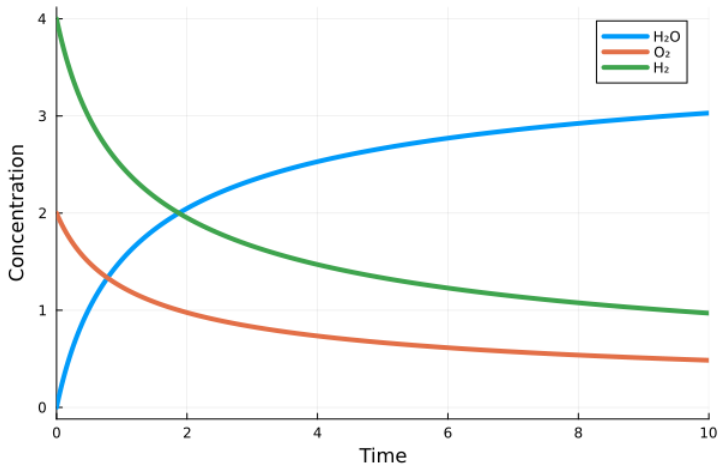


Figure 2.1: Concentration profile of the water reaction: $0.1, 2\,H_2 + O_2 \longrightarrow 2\,H_2O$

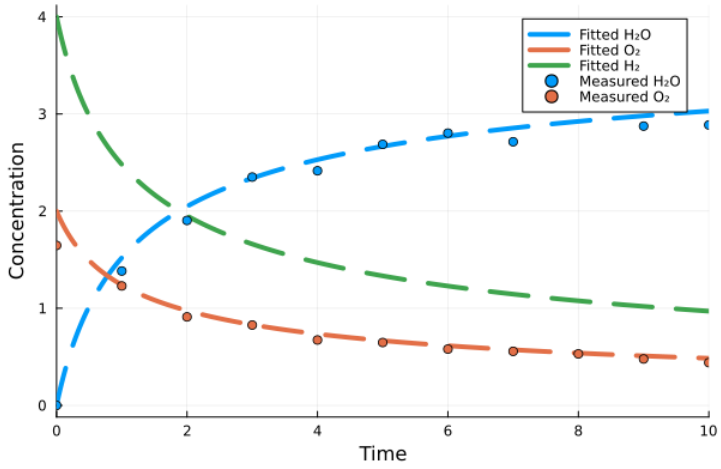


Figure 2.2: Measurements denoted as dots and a simulation with fitted parameter optimization results displayed as a dashed line.

2.2 Program Synthesis

Program synthesis is the task of automatically generating programs that satisfy a given specification [Gulwani et al., 2017]. There are multiple ways to approach program synthesis, however since this thesis makes use of Herb.jl [3], this section will go more in depth specifically about the functionality of this framework. For more information, the thesis Constraint Propagation in Program Synthesis [Swinkels, 2024] also describes these properties very well.

[3] Herb.jl: https://herb-ai.github.io/

A Program Synthesis Problem requires the following components:

- **Input-output examples**: A set of examples that the target program must satisfy.
- **Grammar**: A formal specification of the program search space.
- **Constraints**: Additional conditions that restrict the program space.
- **Search procedure**: An algorithm that explores the grammar-defined space.

Consider for instance the task of synthesizing a simple arithmetic function that increments a variable x by 1. Firstly, we can describe this behaviour as a small set of input-output examples, such as $[1 \mapsto 2, 10 \mapsto 11]$. Next, a grammar is needed that specifies a search space of programs, including a program that increments its input. A grammar consists of rules in the form of $type = expansion$, where a expansion can be an end point or contain reference to types to be expanded further. The example grammar 2.1 includes operations like addition and multiplication, as well as constant values and the variable x.

```
[1]  Number  =  Number * Number
[2]  Number  =  Number + Number
[3]  Number  =  1
[4]  Number  =  2
[5]  Number  =  x
```

Listing 2.1: Example grammar for increment function synthesis

Since the variable x is expected to appear at least once in the program (as it is part of the input-output example), a constraint can be added that ensures rule 5 will be included in every candidate program.

With these components in place, an iterator can now search through all possible programs, evaluating them against the input-output examples. The synthesis process continues until one or more programs are found that satisfy all examples.

In this case, the synthesizer will eventually return the expression:

$$x + 1$$

which satisfies both $1 \mapsto 2$ and $10 \mapsto 11$, and respects the constraint that x must be present in the program.

2.2.1 Grammar

The grammar defines the search space of the synthesizer, the set of all program candidates that can be generated. It consists of a collection of rules that describe how programs can be built.

Each grammar rule has the following properties:

- **Type**: Every rule is associated with a type, specified on the left-hand side of the rule using the syntax `Type = Expansion`. A rule can only be expanded into other rules that match its type.

- **Terminal vs. Non-terminal**: Rules are either terminal or non-terminal. Terminal rules cannot be expanded further. Non-terminal rules reference other rules that will expand further until terminal nodes are reached.

```
[1]   Number    = Number Operation Number
[2:4] Number    = 1 | 2 | x
[5:6] Operation = + | *
```

Listing 2.2: Example grammar for increment function synthesis with multiple types

Consider the following grammar 2.2 that defines simple arithmetic expressions similar to the previous grammar, however now using multiple types. In this example:

- Rules are grouped using the | (or) operator. For example, instead of writing individual rules for each constant or variable, the second line represents three rules.

- The first rule is non-terminal: it defines how a new `Number` expression can be formed by combining two `Number` expressions with an `Operation`.

- The other rules are all terminals: they either define constant values, a variable or operators that can appear in an expression.

In Herb.jl, a program is usually represented using a tree structure, where each node corresponds to a grammar rule and its children correspond to its expansions. For example, the program $x + 1$ can be generated from the rule `Number = Number Operator Number` with its children being a x, $+$ and 1 respectively as shown in figure 2.3.

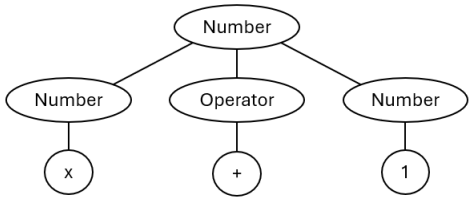


Figure 2.3: Tree example of a program $x + 1$

2.2.2 Search

Herb.jl uses tree-based representations to explore the search space. Each tree corresponds to a set of candidate programs, and the synthesis process involves constructing and refining these trees based on the grammar and constraints.

There are two types of trees in Herb.jl: generic and uniform trees.

- **Generic Tree**: A tree that contains nodes that have a non-terminal type, but are not expanded with children nodes yet.
- **Uniform Tree**: A tree where the structure is fixed: every node has a determined number of children, and the shape is now constant.

At the start of a search procedure, as shown in figure 2.4, a root node will be created with a domain of all grammar rules. Next, the solver will partition the domain into different groups based on the rule types and child structure. For instance, with the previous grammar 2.2, there are three different groups. One with only rule 1, since there is no other rule that expanses into tree children. One with rules 2-4 and another with rules 5 and 6, since these are two groups with their own different types. Since the second tree and third trees are now uniform, the iterator will construct a uniform tree for them. These uniform trees will then be exhaustively used to extract all possible programs that satisfy the constraints. The generic tree will try to expand its children and start again with partitioning their domains into different groups.

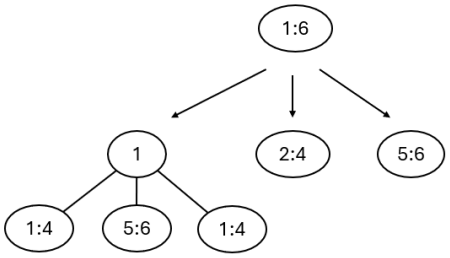


Figure 2.4: Tree partitioning

2.2.3 Constraints

Searching all the possibilities exhaustively has an extremely large search space. To make a problem feasible, good constraints have to be applied. In Herb.jl, constraints describe properties that a (partial) tree should adhere to.

For instance, in the increment problem, a constraint might be to require that the candidate program includes the variable x. To make sure that this rule is in the candidate tree, we can influence the options for each node during the search. For example, when constructing the generic tree, during each decision we will go

through the tree and make sure that there are nodes in the tree that either contain the variable x or can expand to contain this required rule. If there are multiple nodes that still have to be expanded, the propagation cannot make a deduction yet about which (or both) of those nodes will contain the variable rule, however when there is only one hole and the variable x is not anywhere else in the tree yet, this hole can be restricted to only allow rules that will satisfy the constraint. For example, considering the tree partitioning described before in figure 2.4, the first domain will only be split in a node with rule 1 and a node with rule 4 (the variable x) since the other options will never be able to result in a candidate program that contains the variable x.

If the iterator has an uniform tree, the constraint still works mostly the same, however note that the structure of the uniform tree does not change any more, so the constraint can keep information stored. For example, by keeping track of the position of the available holes (nodes with multiple possible rules), the whole tree does not need to be searched again.

Constraints are not limited to the full tree, each node can get its own local constraints with its own local logic defined. For example, the $+$ and $*$ operators are commutative, and thus we could add a constraint that will prevent candidate programs that will have the same output. To implement this, we can add a local constraint to each node with rule 1 and enforce that its children nodes with type Number are ordered: For example, the right Number rule is greater or equal to the left Number rule. Thus preventing the candidate $2 + 1$ from ever being attempted.

Chapter 3

Related work

The automated discovery of Chemical Reaction Networks (CRNs) has received growing interest across computational chemistry, systems biology, and machine learning. In this chapter, we discuss some notable approaches for CRN discovery.

3.1 Simplest Mechanism Builder Algorithm

The Simplest Mechanism Builder Algorithm (SiMBA) [Ángel de Carvalho Servia et al., 2025] is designed for automated microkinetic model discovery from kinetic data, without requiring complete knowledge. Microkinetic models describe the same problems as CRN. SiMBA employs a search procedure that incrementally increases the complexity of the model and performs model evaluation via parameter estimation.

While SiMBA effectively predicts the presence of intermediates, it does not attempt to discover the identity of unknown molecules. Necessitating expert input to contextualize and refine the discovered network. In contrast, our approach explores a discrete program space of chemical reaction networks, aiming to also recover unknown molecules directly from data.

3.2 Automated Discovery of Kinetic Rate Models

This research proposes two methodological frameworks, ADoK-S and ADoK-W [de Carvalho Servia et al., 2024], for the automated generation of kinetic models from experimental concentration data. Both frameworks leverage genetic programming for model generation and a sequential optimization step for model refinement. ADoK-S employs a conventional approach, necessitating estimated rate measurements for deriving kinetic rate models. In contrast, ADoK-W applies a weak formulation of symbolic regression by directly constructing rate models from measured concentration data through an embedded integration step, thereby bypassing the rate estimation step entirely.

3.3 Nested Evolutionary Algorithm for CRN Design

The research [Degrand et al., 2019] proposes an method for constructing Chemical Reaction Networks based on evolution algorithms. It utilizes a nested search algorithm, where a genetic algorithm evolves the CRN structure and a parameter optimization function optimizes its kinetic parameters for the CRNs. At each generation, mutations to the CRNs are made to search for better options. This approach aims to discover CRNs without requiring prior knowledge of their structure.

3.4 Syntax-Guided Synthesis for CRNs

This paper [Cardelli et al., 2017] addresses the problem of finding CRNs by syntax-guided synthesis. It introduces a sketching language that contains syntactic constraints that allows 'holes' for unknown species, rates, or stoichiometric constants. The goal is to find a CRN that satisfies specified behaviour and minimizes a defined cost function. The approach encodes the synthesis problem as a Satisfiability Modulo Theories (SMT) problem.

Chapter 4

Problem Definition

The main objective of this thesis is to use program synthesis to automatically construct complete chemical reaction networks that explain experimental observations or a given set of requirements given for the CRN. In many practical scenarios, only partial observations of the system are available. The challenge is to infer the underlying reaction network that could plausibly produce this behaviour.

More formally, the problem can be stated as follows:

Given:

- A set of observed molecules $M = \{m_1, \ldots, m_n\}$.
- Corresponding concentration measurements over time $C_i(t)$ for each $m_i \in M$.

Find:

- A set of additional (unobserved) molecules $M' = \{m_{n+1}, \ldots\}$.
- A set of reactions R that define the interaction among the molecules in $M \cup M'$.
- A chemical reaction network N, as defined in section 2.1, such that simulations of N reproduce the observed concentration profiles $C_i(t)$ for all $m_i \in M$.

4.1 Example Problem

To illustrate the problem, consider an esterification reaction for which only the concentrations of the molecules $C_3H_6O_2$, $C_2H_4O_2$, and C_2H_6O have been measured. These observed molecules represent the set M and their concentration measurements, shown in figure 4.1 as solid lines, represent $C_i(t)$. Now the goal for the solver is to find the full underlying chemical reaction network N, consisting of two reactions R, complete with the missing molecules $M' = \{CH_2O_2, CH_4O, H_2O\}$:

$$0.2, CH_2O_2 + C_2H_6O \longrightarrow C_3H_6O_2 + H_2O$$
$$0.1, C_3H_6O_2 + CH_4O \longrightarrow C_2H_4O_2 + C_2H_6O$$

Using this synthesized network, the concentration profiles of the unmeasured molecules can also be found, as shown with the dashed lines in figure 4.1

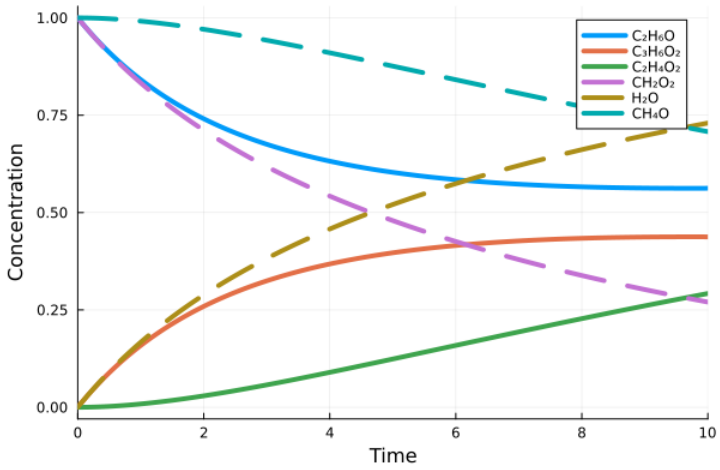


Figure 4.1: Example esterification problem with in solid lines the measured molecules and in dashed lines the unobserved molecules

Chapter 5

Methods

This chapter describes the methodology developed to solve the problem outlined in Chapter 4. The approach consists of the following stages:

- **Analyse the Problem:** Given a problem definition, all available properties are gathered. For example, from analysing the measured concentration profiles of the molecules, we can gather whether a molecule should be the input or output of a reaction.
- **Synthesize CRN candidates:** Using the available information, complete CRN candidates are synthesized. This can be done by creating a single synthesis problem or by splitting the problem up into different stages for synthesizing molecules, reactions, and networks separately. Each option requires a different combination of constraints and grammar that will be explained more in the next sections.
- **Evaluate the Networks:** Given the synthesis step results in many possible candidates, a ranking is required that defines properties that are more desirable than others.

5.1 Problem Analysis

The first step is to analyse the given problem, which contains information about the molecules that are observed and their behaviour over time. It is assumed that all the measured species are at least part of the target CRN. Furthermore, looking at the corresponding concentration graphs for the measured species, increasing or decreasing trends can be observed. This tells us if a species is required to be on the input or output side of a reaction. For instance, when looking at figure 5.1, the molecule H_2O can be seen to be increasing, thus we know that there should be at least a single reaction in the CRN where that molecule is on the right side. Note that a concentration graph could also, for instance, be a bell curve, meaning that the species is both increasing and decreasing at some point. Thus, the CRN should

have a reaction with the species on the right side and another one with the species on the left side. Allowing us to reduce the search space, since we know where the measured species are located in the reactions.

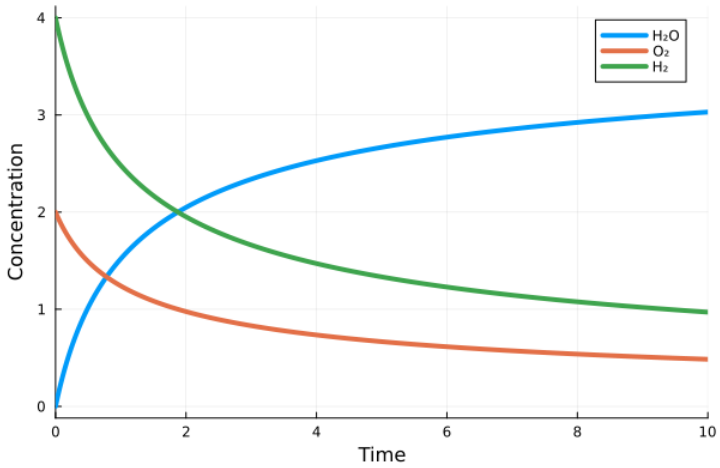


Figure 5.1: Concentration profiles for the following reaction: $2H_2 + O_2 \rightarrow 2H_2O$

Furthermore, lists of possible molecules or reactions can also be passed on, which are not necessarily measured in the CRN, but might be likely options. This provided list of molecules or reactions can then be used by the solver to add constraints to make sure that they are indeed in the reaction network, again reducing the search space.

5.2 Molecule Synthesiser

Some of the molecules that are needed to synthesize the target CRN might not be known from the problem definition. Therefore, new molecules need to be synthesized as part of the search process. This section will discuss the grammar and constraints used to synthesize molecules.

5.2.1 Grammar

The grammar for the molecule synthesis is based on the OpenSMILES specification [1]. However, to have more control over the generated structures, the grammar for our synthesizer requires all atoms to be explicitly specified. Which is denoted by all the atoms in brackets. Furthermore, our grammar currently does not support isotopes and charges, since in the proposed experiments this is not required. Furthermore, some changes have been made to implicitly restrict some invalid outputs. For instance, the OpenSMILES specification could create SMILES strings ending in a branch, which the grammar below does not.

[1] OpenSMILES specification: http://opensmiles.org/

Terminal rules may be changed to suit the specific problem. Based on the all the molecules gathered from the problem description, a set with all the atoms is collected from which molecules structures can be constructed. In the example below, the grammar is demonstrated with a four atoms and a maximum bond order of three:

```
[1]  molecule   =  chain

[2]  chain      =  atom ringbonds
[3]  chain      =  structure bond chain

[4]  structure  =  atom ringbonds branches

[5]  branch     =  ( bond chain )
[6]  branches   =  nothing
[7]  branches   =  branch branches

[8]  ringbond   =  bond digit
[9]  ringbonds  =  nothing
[10] ringbonds  =  ringbond ringbonds

[11] digit      =  1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9
[12] bond       =  - | = | ≡
[13] atom       =  [H] | [O] | [N] | [C]
```

5.2.2 Constraints

The grammar described previously still requires additional constraints to ensure that the generated molecules are valid. However, distilling the requirements to create a chemically possible molecule into a few concrete constraints is more challenging than expected. Therefore, the current implementation only enforces that each atom has the expected amount of bonds and that all the ringbonds are closed. For example, a [H] atom must form exactly one bond, while an [O] atom must form two. And a specific ringbond digit always has a corresponding closing connection.

Generic Atom

Since the generic tree can still further develop, the final structure is unknown. Nevertheless, we can still make some deductions. The `GenericAtom` constraint is posted on any node with the molecule type. Then, during propagation, it will gather all the bond nodes while going down the tree and if an atom is found, restrict the search space accordingly.

For example, if we find an atom node during propagation and we have determined that there are two bond nodes that connect to this atom, we can infer that the

atom should at least be able to have two connections. Furthermore, if, for example the atom can have a maximum of 2 connections, we can also infer that for each of the connected bond nodes, it cannot be more than a single bond. Using this information, we can restrict the possibilities on the atom and bond nodes.

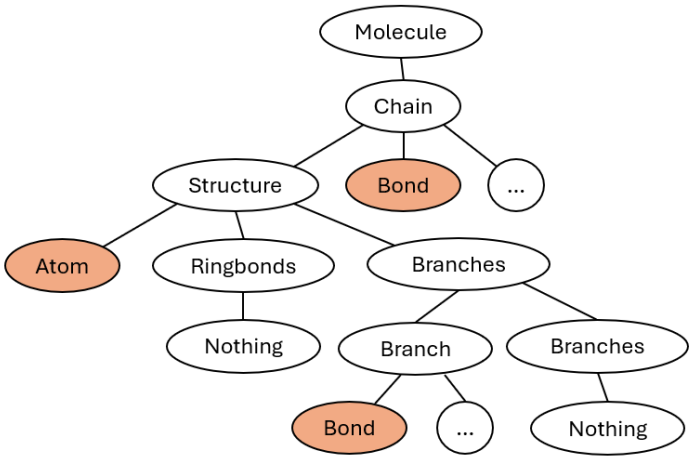


Figure 5.2: Partial tree with marked nodes that are constrained by the Atom Constraints

In the instance shown in figure 5.2, an atom with corresponding bonds is highlighted in orange. In this case the atom can be restricted such that it cannot be a hydrogen atom. And assuming the atom would be an oxygen atom, the bonds would both need to be a single connection.

Since the tree is not uniform yet, we also have to take holes into account that can still expand into more bonds. If such a hole is found, we can infer that there is at least a single bond in that expansion chain, however, for further information the propagation will have to wait until the node has branched.

Uniform Atom

Once the tree becomes uniform, the structure of the tree will not change anymore. Therefore, the constraints can store information about the structure. Instead of a single constraint at the node with type molecule, now all of the nodes with type atom will get their own `UniformAtom` constraint imposed. Now, instead of searching for all relevant bonds during each time the constraint is checked, this `UniformAtom` constraint can gather the locations of all bond nodes that are connected to the atom only once upon construction of the uniform tree. When propagating, similarly to the generic atom constraint, the maximum and minimum values of the bonds and atom, will both be checked and updated accordingly.

Uniform Ringbond

The UniformRingbond constraint imposes the following rules.

1. It prevents duplicate molecules that have the same structure with only the ring number altered. For example, the molecules `[O]-1-[O]-[O]-1` and `[O]-2-[O]-[O]-2` would result in the the same structure.

2. It makes sure that each used digit will be in the molecule twice to complete the ringbond.

3. It will make sure that ringbonds do not connect two molecules that are already connected to each other or to itself.

Restricting duplicate molecules is done by having the max ringbond be dependent on the position of the ringbond. For example, the leftmost ringbond node can only take the value 1, the second leftmost can take the values 1 or 2, and this continues to increase until the last item. Also, if a single atom has multiple ringbond digits, these should be incrementing, since changing the order would create identical molecules.

To make sure that ringbonds do not connect to themselves or have multiple ringbonds between the same two atoms, the notion of ringbond groups is introduced. When constructing the constraint, the ringbonds are grouped based on where they are connected two. During the propagation, uniqueness in these groups is imposed.

Take, for example, a molecule with four ringbond positions that need to be assigned ring digits: `[C]-`r_1`≡[C]-[C]-`r_2`-`r_3`-[C]≡[C]-`r_4. The first step of the constraint is to limit the possibilities incremented by the position, thus $r_1 \in \{1\}$, $r_2 \in \{1, 2\}$, $r_3 \in \{1, 2, 3\}$ and $r_4 \in \{1, 2, 3, 4\}$. Since each digit should be twice in the molecule and there are a total of 4 places, we also know that the limit should be 2, thus $r_1 \in \{1\}$, $r_2 \in \{1, 2\}$, $r_3 \in \{1, 2\}$ and $r_4 \in \{1, 2\}$. By making sure that ringbonds do not connect, we can also infer that $r_2 \neq r_3$. Finally, the second and third digit should be increasing to prevent duplicates, $r_2 < r_3$. By combining these constraints, the only possible solution left is $r_1 = 1$, $r_2 = 1$, $r_3 = 2$ and $r_4 = 2$: `[C]-1≡[C]-[C]-1-2-[C]≡[C]-2`

5.3 Reaction Synthesiser

The synthesis of reactions can be performed in two ways. Either by starting from a predefined list of molecules or by combining the molecule grammar with the reaction grammar, enabling the construction of complete reactions within a single synthesis step.

5.3.1 Grammars

The reaction grammar consists of two lists of molecules, one for the inputs and one for the outputs. In the example below the molecule rule is terminated with for ex-

ample pre-generated or given molecules. Note that this rule could also be replaced with the full molecule grammar, thus creating a single iterator that synthesises reactions.

```
[1]   reaction       =  molecule_list → molecule_list

[2]   molecule_list  =  molecule + molecule_list
[3]   molecule_list  =  nothing

[4-6]  molecule       =  H2O | O2 | H2
```

Rule [1] defines a reaction as a list of input molecules and a list of output molecules, separated by a reaction arrow. The molecules in rule [4] can either be a predefined list if synthesized earlier or be replaced with the molecule grammar described in section 5.2.

Rules [2] and [3] represent the tail-ended molecule lists. Note that since chemical reactions do not have a specific ordering required, therefore an ordering constraint is later applied to reduce symmetry duplicates.

When combining this grammar with the molecule grammar, each molecule node can itself expand into a valid SMILES-like tree structure, resulting in a single program synthesis problem.

5.3.2 Constraints

There are two main constraints that are imposed upon the reaction grammar. Firstly, we will make sure that the reactions are atom balances. Secondly, we need a constraint to restrict the duplicates generated due to symmetry. Furthermore, there are some simpler constraints that reduce the amount candidates that will be rejected later in the process. For example, a forbidden constraint that makes sure that the reaction is not the same at both sides and a forbidden constraint that makes sure the molecule list at least contains a single molecule. However, those will not be explained in depth.

Balanced Reaction

A reaction should have the same amounts of atoms in its input as its output. Therefore, multiple constraints are in place.

Firstly, in the case when the molecules are predefined by a previous synthesizer, atom balancing is done for each uniform tree found. Since it is quite difficult to restrict possibilities in a situation where there are multiple holes in a reaction, the choice has been made to pre calculate the intersection set of possible reactions.

For both of the reaction sides, all possible combinations of rules are gathered and translated to their corresponding atom counts. Then by taking the intersection of these atom count sets, we get all the atom balanced reactions. This makes it then possible to restrict the uniform tree with the rules that contain in this possibility

set. While this does not add much functionality in the case of a synthesizer that only synthesizes reactions based on a set of molecules, it does currently improve the performance when you add more constraints in a synthesizer that does multiple steps.

When the Grammar is extended with the molecule grammar, the Balanced Reaction functions a bit different. Firstly, when the structure of the tree is not yet uniform, it will make sure that the amount of atom holes is equal on both sides of the reaction. Then, when a uniform tree is constructed, the propagation restricts the possible atoms based on the current fixed holes until either the tree proves infeasible or a solution is found.

Ordered List

In a chemical reaction network, the order of the molecules on a side of the reaction does not influence the outcome. For example, the reaction $2H_2 + O_2 \rightarrow 2H_2O$ is the same as $O_2 + 2H_2 \rightarrow 2H_2O$. To restrict these duplicate options, we introduce an order constraint on the molecule list. Due to how the reaction grammar is constructed, implementing this constraint can be done by adding a small local constraint on each node with rule type 2. If its right child node is also of type 2, then between their respective molecules a comparison takes place. The lower molecule list should have rules that are the same or higher as the higher molecule list as seen in figure 5.3.

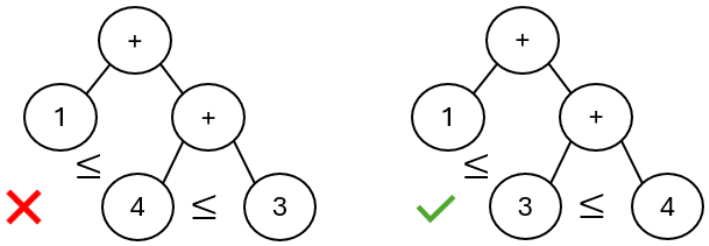


Figure 5.3: Tail ended ordered list

5.4 Network Synthesiser

With molecules and reactions setup, the final step is to construct the complete chemical reaction networks.

5.4.1 Grammars

The grammar for synthesizing a chemical reaction network is similar to the reaction network. It is defined as a tail ended list of reactions, where each reaction

must be unique within the network. This grammar can again be given a list of synthesized reactions or combined with the grammars and constraints of the previous two sections to synthesize full networks with a single iterator. An example of the grammar is given below:

```
[1]   network        =  reaction_list

[2]   reaction_list  =  reaction + reaction_list
[3]   reaction_list  =  nothing

[4-5]  reaction       =  2H2 + O2 → 2H2O |
                         CH4 + 2O2 → 2H2O
```

5.4.2 Constraints

Similar to the molecule list, this grammar again makes use of the ordered list. A forbidden constraint is added to ensure that a reaction list cannot have multiple reactions that are the same.

Contains Molecules

From analysing the problem definition, like mentioned in section 5.1, a set of required molecules is already known with their expected place in the CRN. To incorporate this information, the synthesis process should only generate networks that include all of these required molecules.

If the grammar is constructed using complete reactions, the constraint will make sure that the reaction list will not have an ending rule for the list (rule 3) until all the required molecules are in the list.

When using the combined reaction and network grammar, the reaction grammar expands with a separate rule that just contains required molecules. The reaction grammar also posts a constraint that will make sure that the network contains enough nodes with the rule type "required molecule" as shown below. When propagating over the complete tree, the constraint makes sure that all the required molecules are at the right place before the reaction list is ended.

```
[5]   molecule_list      =  required_molecule + molecule_list
[6]   required_molecule  =  H2O | O2 | H2
```

5.5 Synthesizer Pipelines

As mentioned in the previous sections, the grammars for the synthesizers can be combined in multiple ways to create different pipelines. The following pipelines are tested:

1. Problem Definition → Molecules → Reactions → Networks
2. Problem Definition → Molecules → Networks
3. Problem Definition → Reactions → Networks
4. Problem Definition → Networks

Each of these pipelines reflects a different level of synthesis modularity. Pipeline 1 has the most synthesizer steps, while Pipeline 4 has a single end-to-end synthesis step.

While a single synthesis step is arguably the most elegant solution, this option might contain duplicate searches depending on the underlying algorithm.

Each of the staged pipelines follows a similar principle of incrementing the candidate set of the previous options by one, then adding the synthesized candidate as a `Contains` constraint to the next option and iterating the subsequent synthesizer until exhaustion. For example, pipeline 1 has the following flow as shown in figure 5.4.

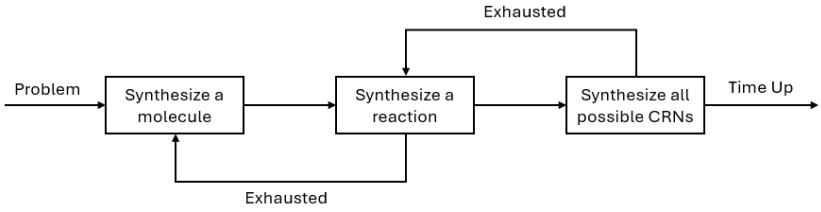


Figure 5.4: Staged Pipeline

The pipeline starts with the problem analysis, after which a molecule synthesizer is created. Then, for every molecule created by the molecule synthesizer, a reaction synthesizer is created with all the collected molecules, and the newly synthesized molecule is added as a required molecule. This makes sure that the synthesized reactions are different from previous generated candidates. With the list of reactions, a network synthesizer is created, which will generate the final candidates that will be used in scoring. Again, this will have the collected list of reactions, and the latest reaction as a requirement. This process will continue until either a timeout or complete exhaustion of the molecules occurs.

5.6 CRN evaluation

Synthesized candidate networks are not all equally useful. A scoring function is used to evaluate and rank them based on a set of properties. The scoring function does not guarantee the correct solution, but helps prioritize the most likely candidates.

The evaluation score for a network is the combination of the following properties:

- **Profile matching:** The most important score is based on how well the generated network reproduces the expected concentration profiles of the measured molecules.
- **Minimal change in a single reaction:** Reactions are scored based on the amount changes they cause to the molecules. Reactions that make minimal modifications (e.g., forming or breaking a single bond) are preferred over those that drastically alter molecular structures.
- **Species count:** Networks with fewer unique chemical species are preferred, as this avoids unnecessary intermediates.
- **Reaction count:** Networks with fewer reactions are prioritized. This discourages the inclusion of redundant or non-functional reactions.

The comparison to the expected species profiles is done using a simulator. The network candidate is translated to a system of differential equations in Catalyst.jl and using parameter tuning, parameters that are as close as possible to the expected result are found.

The combination of all these scores is used to finally rank the reaction networks and the best networks are returned back to the user.

Chapter 6

Results and Discussion

This chapter presents experiments designed to address the following research questions:

- Do the constraints successfully reduce the search space?
- Is the solver able to find the correct chemical reaction network?
- Is the expected solution one of the best scored options?

The sections below begin by detailing the experimental setup, followed by an analysis of the search space reduction, feasibility, and solution accuracy.

6.1 Experimental Setup

The experiments are run on a system with an AMD 7950x and 128GB of ram using the code from the CRNSynthesizer repository (v0.1.0). A timeout of 10 minutes is used per experiment. To evaluate different methods, this thesis considers CRNs with varying levels of difficulty. The target CRNs used are described in the following sections.

6.1.1 Water Reaction

The first test case is a simple CRN representing water formation. This example provides a minimal baseline to verify whether the synthesis pipeline can recover an expected CRN. Only the concentration profiles of O_2 and H_2O are assumed to be measured, meaning that the solver will have to find that the H_2 molecule is also part of the CRN and finally has to find the following CRN:

$$\mathrm{r}_1, 2\,\mathrm{H}_2 + \mathrm{O}_2 \longrightarrow 2\,\mathrm{H}_2\mathrm{O}$$

The correct molecules are:

- H_2: [H]-[H]
- O_2: [O]=[O]
- H_2O: [H]-[O]-[H]

6.1.2 Methane Combustion

Secondly, an experiment is set up for methane combustion. This experiment assumes that only the methane and water molecules are known to take part in the network. Therefore, the solver will have to synthesize the two molecules, oxygen and carbon dioxide.

$$\mathrm{r}_1, \mathrm{CH_4} + 2\,\mathrm{O_2} \longrightarrow \mathrm{CO_2} + 2\,\mathrm{H_2O}$$

The correct molecules are:

- CH_4: [H]-[C](-[H])(-[H])-[H]
- O_2: [O]=[O]
- H_2O: [H]-[O]-[H]
- CO_2: [O]=[C]=[O]

6.1.3 Ethylene glycol

Next, an experiment is setup that requires a molecule with a ringbond to be synthesized, namely ethylene oxide (C_2H_4O). We assume that H_2O and $C_2H_6O_2$ are measured.

$$\mathrm{r}_1, \mathrm{C_2H_4O} + \mathrm{H_2O} \longrightarrow \mathrm{C_2H_6O_2}$$

The correct molecules are:

- C_2H_4O: [H]-[C](-[O]-1)(-[H])-[C]-1(-[H])-[H]
- $C_2H_6O_2$: [H]-[O]-[C](-[H])(-[H])-[C](-[H])(-[H])-[O]-[H]
- H_2O: [H]-[O]-[H]

6.1.4 Esterification Reaction

The following esterification process is a more complex test case involving two reactions. For this chemical reaction network, we consider that only the concentration profiles of the molecules $C_3H_6O_2$, $C_2H_4O_2$ and C_2H_6O are known.

$$\mathrm{r}_1, \mathrm{CH_2O_2} + \mathrm{C_2H_6O} \longrightarrow \mathrm{C_3H_6O_2} + \mathrm{H_2O}$$
$$\mathrm{r}_2, \mathrm{C_3H_6O_2} + \mathrm{CH_4O} \longrightarrow \mathrm{C_2H_4O_2} + \mathrm{C_2H_6O}$$

The correct molecules are:

- CH_2O_2: [H]-[C](=[O])-[O]-[H]
- C_2H_6O: [C](-[C](-[H])(-[H])-[O]-[H])(-[H])(-[H])-[H]
- $C_3H_6O_2$: [H]-[C](=[O])-[O]-[C](-[H])(-[H])-[C](-[H])(-[H])-[H]
- H_2O: [H]-[O]-[H]
- CH_4O: [C](-[H])(-[H])(-[H])-[O]-[H]
- $C_2H_4O_2$: [C](-[H])(=[O])-[O]-[C](-[H])(-[H])-[H]

6.1.5 Synthesizer steps

As described in the methods chapter, the task of synthesizing a complete Chemical Reaction Network can be split into three different subproblems.

- **Molecule Synthesis:** Given the input problem and any measured molecules, generate plausible candidates for missing species.
- **Reaction Synthesis:** Given the known and synthesized molecules, construct reactions that are atom balanced.
- **Network Synthesis:** Finally, using the synthesized reactions, combine them into a CRN that produces simulation outputs that match the observed concentration profiles.

These synthesizer steps have their own constraints and they can be combined into various pipelines described in section 5.5. Each of these steps will be separately evaluated.

6.2 Search Space Reduction

The search space for the CRNs is enormous, that is why good constraints are very important. To evaluate the impact of the constraints proposed, the total number of program candidates has been tested with and without constraints until exhaustion of the iterator at a certain depth. This will give an indication on how much of the search space was marked unfeasible due to the constraint. Furthermore, the runtime is also tracked, to ensure that the overhead introduced by constraints does not outweigh their performance benefits from reducing the search space. To evaluate the constraints, the synthesizer steps have been tested on the methane problem with various constraints turned on and off. The expected unique candidates are based on a manual check on the generated constraints.

6.2.1 Molecule Synthesis

The implementation of the molecule grammar constraints described in the methods is all built into a single constraint called `ValidSmiles`. To see how it influences the search space, a comparison is made between a run with and without the constraint.

Setting	Candidates	Time (s)
Expected Unique Molecules	7	-
With ValidSmiles	7	0.002
Without Constraint	18399	0.139

Table 6.1: Molecule Synthesis with depth 6 given the methane problem

The methane problem with a depth of 6 resulted in table 6.1. Where we can see that both the amount of candidates and the runtime have significantly decreased when the constraint is applied.

Setting	Candidates	Time (s)
Expected Unique Molecules	22	-
With ValidSmiles	33	0.031
Without Constraint	DNF	DNF

Table 6.2: Molecule Synthesis with depth 7 given the methane problem

However, when setting the depth to 7, table 6.2 shows that there are 33 candidate molecules returned while only 22 of these are unique. Meaning that there are still some symmetries that are not fully constrained. While it is difficult to compare how much the search space is reduced in this test, the fact that without constraints the synthesizer does not finish within the time limit, suggest the constraint is significantly reducing the search space.

6.2.2 Reaction Synthesis

There are two synthesizer options for the creation of reactions. A synthesizer that starts with molecules synthesized in a previous step and a synthesizer that starts with atoms. For both of these synthesizers the effect of the `Ordered` and `BalancedReaction` constraints have been tested.

Setting	Candidates	Time (s)
Expected Unique Reactions	6	-
All Constraints	6	0.924
Without BalancedReaction	870	1.455
Without Ordered	136	0.916
Without BalancedReaction and Ordered	8190	2.130

Table 6.3: Reaction Synthesis from Atoms with depth 8 given the methane problem

In table 6.3, the results of a reaction synthesis starting with atoms is displayed. Which shows that the constraints do successfully decrease the search space. Although the ordered constraint did not seem to reduce the runtime, the reduced number of candidates while keeping the synthesis runtime similar, could still improve the overall performance of the solver due to not needing to evaluate as many candidate programs later in the process.

For the reaction synthesizer from molecules, 10 molecules have been given as a starting point for the synthesizer to use. In table 6.4 the first indications can be seen that the staged pipeline has significant performance benefits as the staged pipeline is generating 160 candidates in less time than the combined synthesizer takes to generate 6 candidates. This is likely because staging the pro-

Setting	Candidates	Time (s)
Expected Unique Reactions	160	-
All Constraints	160	0.016
Without BalancedReaction	80940	2.779
Without Ordered	160	0.014
Without BalancedReaction and Ordered	1230990	45.797

Table 6.4: Reaction Synthesizer from 10 Molecules with depth 5 given the methane problem

cess in this way can make sure that molecules do not have to be rediscovered for each reaction and that both the molecules used in the construction of reactions are unique. The implementation of the `BalancedReaction` constraint already enforces the order of the molecules. The logic of the `ordered` was added because the constraint is much simpler and it might prune options before the more complex `BalancedReaction` constraint logic has to trigger. However, the runtime without the ordered constraint seems to be faster, suggesting that this currently is not effective.

6.2.3 Network Synthesis

There are three ways to create the networks. Firstly, by merging all three grammars into a single synthesizer step. Secondly, by using a list of generated molecules and finally from a list of generated reactions.

Integrating all three stages into a single synthesizer appears conceptually appealing, since there is no need to set separate settings and limits for each stage. However, in practice it led to very slow performance. For example, running until exhaustion would never terminate before the time limit. Therefore, only the search space of the staged synthesizers have been evaluated further.

Setting	Candidates	Time (s)
Expected Unique Networks	417	-
All Constraints	417	7.332
Without ContainsMolecules	5142	7.556
Without Ordered	667	7.299
Without ContainsMolecules and Ordered	8623	10.023

Table 6.5: Network Synthesizer with depth 8 from 10 Molecules given the methane problem

If we look at the results from the network synthesizer starting from molecules in table 6.5, we can see that many options are generated. Interestingly, the runtime improvements of this constraint are not as significant as you would expect from a constraint, suggesting there is still room for improvement. Despite no improve-

ment in time in this step, evaluating just 417 CRNs will cost less runtime when simulating the networks later in the solver process.

Setting	Candidates	Time (s)
Expected Unique Networks	96	-
All Constraints	96	0.014
Without ContainsMolecules	20101	0.035
Without Ordered	192	0.018
Without ContainsMolecules and Ordered	40001	0.059

Table 6.6: Network Synthesizer with depth 4 from 200 Reactions given the methane problem

Finally, the table 6.6 shows the performance of the constraints on the synthesizer step from reactions to networks. In this table, the importance of the problem analysis is visible. By applying the `ContainsMolecules` constraint, which uses the information from the analysis, the search space is reduced from 20101 candidates to only 96 candidates.

6.3 Feasibility

While the constraints do prune a large part of the program space, there are still many candidate programs that do not match the target network. For the feasibility test, the stopping condition for all synthesizer steps will be when it finds the needed candidates. For instance, with the water reaction, the molecule synthesis stage will stop if it finds the missing O_2 molecule and the reaction stage will stop if the reaction $2\,H_2 + O_2 \longrightarrow 2\,H_2O$ is found.

Pipeline	Time Until Found	Candidates Generated
Problem → Networks	DNF	DNF
Problem → Molecules → Networks	0.049	1
Problem → Reactions → Networks	502.236	2
Problem → Molecules → Reactions → Networks	0.007	2

Table 6.7: Feasibility of the water reaction

Table 6.7 shows a significant performance difference between the pipelines. Staging the pipeline with a molecule synthesis step finds the network within the first second. This is likely due to that, when generating a reaction without the in between step, each molecule in every reaction must be synthesized independently. While the staged pipeline only has to synthesize each molecule once.

As shown in Table 6.8, similar trends are observed for the methane combustion experiment. The fully combined pipeline and the pipeline that skips molecule synthesis both fail to find the correct network within the time limit. In contrast, the staged pipelines that include the molecule synthesis step succeed. Notably, the

Pipeline	Time Until Found	Candidates Generated
Problem → Networks	DNF	DNF
Problem → Molecules → Networks	0.083	2
Problem → Reactions → Networks	DNF	DNF
Problem → Molecules → Reactions → Networks	0.024	2

Table 6.8: Feasibility of the methane experiment

full three-stage pipeline performs best, finding the correct network in just 0.024 seconds.

Pipeline	Time Until Found	Candidates Generated
Problem → Networks	DNF	DNF
Problem → Molecules → Networks	4.280	3
Problem → Reactions → Networks	DNF	DNF
Problem → Molecules → Reactions → Networks	5.237	3

Table 6.9: Feasibility of the ethylene experiment

The ethylene experiment in table 6.9 similarly shows that only the staged pipelines with the molecule step manage to find the solution in the time limit. However, in contrast to the previous examples, this time the runtime of the fully staged pipeline was significantly longer. This might be due to increased complexity of the missing molecule, resulting in many more molecules and therefore also reactions to synthesize before the correct one is found.

Pipeline	Time Until Found	Candidates Generated
Problem → Networks	DNF	DNF
Problem → Molecules → Networks	DNF	DNF
Problem → Reactions → Networks	DNF	DNF
Problem → Molecules → Reactions → Networks	24.939	2001

Table 6.10: Feasibility of the esterification reaction

For the esterification experiment 6.10, only the fully staged pipeline has been able to find the target network within the time limit. This is likely because this is the only experiment with multiple reactions, allowing the performance improvements of moving the synthesis of the reactions to a separate step to become apparent.

6.4 Accuracy

One of the challenges that comes with generating many options that might match your result is to define what is the best. To differentiate between the candidates, they are individually scored. This test will indicate how high the corresponding target network was ranked. Based on the feasibility experiments, a limit is set on the maximum number of networks.

Pipeline	Total Networks	Target Rank
Problem → Networks	DNF	DNF
Problem → Molecules → Networks	1000	7
Problem → Reactions → Networks	DNF	DNF
Problem → Molecules → Reactions → Networks	1000	3

Table 6.11: Accuracy results for the Water Reaction

In table 6.11 the result of a run can be found with the initial molecules set to 5 and the initial number of reactions to 10. Then using these parameters, the synthesizer pipelines are run until either the time limit of 10 minutes or a limit of 1000 networks has been reached. While the target CRN is not evaluated as the best candidate, it still ranks very high. In both situations in the top 10 of the 1000 candidate CRNs.

Pipeline	Total Networks	Target Rank
Problem → Networks	DNF	DNF
Problem → Molecules → Networks	1000	12
Problem → Reactions → Networks	DNF	DNF
Problem → Molecules → Reactions → Networks	1000	3

Table 6.12: Accuracy results for the methane experiment

Table 6.12 presents the accuracy results for the methane combustion test. Among the staged pipelines, the three-stage pipeline again ranks the correct network highest, placing it 3rd out of 1000 generated candidates. The two-stage pipeline also ranks the correct solution relatively well at position 16. Although there is a notable difference in ranking between the two pipelines, this discrepancy is not due to differences in the underlying search space, since they have the same search space available. Instead, the variation is likely due to the introduction of the reactions step altering the search order.

Pipeline	Total Networks	Target Rank
Problem → Networks	DNF	DNF
Problem → Molecules → Networks	1000	1
Problem → Reactions → Networks	DNF	DNF
Problem → Molecules → Reactions → Networks	1000	1

Table 6.13: Accuracy results for the Ethylene Reaction

Table 6.13 shows the results for the ethylene reaction test. Similar to the other examples, only the pipelines without the molecules step fail to find the target network within the time limit. Interestingly, the target network is ranked the best in both cases. This is likely due to that the missing molecule has a larger structure and that other CRN options would require using more smaller molecules or more structural changes, which are penalized by the evaluation function.

Since for the esterification reaction, the reaction network that has to be synthes-

Pipeline	Total Networks	Target Rank
Problem → Networks	DNF	DNF
Problem → Molecules → Networks	DNF	DNF
Problem → Reactions → Networks	DNF	DNF
Problem → Molecules → Reactions → Networks	3000	13

Table 6.14: Accuracy results for the Esterification Reaction

ized is more complex, the starting conditions for this experiment will be to start with 70 molecules and 56000 reactions. Also, based on the feasibility benchmark, the search is limited to 3000 networks. As seen in table 6.14 the target network is in position 13 out of these 3000 candidates. This shows that for even a more complex network, the solver is able to identify the target network.

Chapter 7

Conclusions and Future Work

This thesis explored the use of program synthesis techniques for the construction of Chemical Reaction Networks (CRNs) from incomplete or partial concentration data. This was done by first making a formalization of CRN discovery as a program synthesis problem using grammars and constraints. Next, these grammars and constraints were combined into different modular solver pipelines capable of synthesizing molecules, reactions, and entire chemical reaction networks from (partial) data. Lastly, an evaluation framework was created to test solver feasibility, performance, and accuracy on multiple benchmark problems.

The results demonstrate that the solver can reliably generate valid CRNs for problems of varying difficulty, including the esterification problem, which involves synthesizing a network with three unknown molecules and two reactions. Across all stages of the synthesis pipeline, the introduction of chemical constraints substantially reduced the search space. For example, in the methane experiment, applying the `ValidSmiles` constraint alone decreased the number of candidate molecules from over 18000 to just 7, significantly improving solver efficiency without sacrificing correctness.

Secondly, the addition of constraints did not only reduce the number of candidates, but also consistently reduced the total runtime of the solver. Meaning that the addition of constraints did not introduce more computational overhead than the time saved by pruning the search space.

Moreover, decomposing the synthesis task into separate stages consistently led to faster runtimes and higher feasibility compared to a single end-to-end synthesis approach. In fact, the fully integrated approach failed to terminate within the allowed time limit across all test cases, while the staged pipeline often returned valid solutions in under a second.

Finally, the solver was not only able to synthesize the correct CRNs for each benchmark, but also consistently ranked them near the top. Demonstrating the effectiveness of the scoring function in aligning synthesized network behaviour with expected concentration profiles.

7.1 Future Work

While the current implementation provides a good foundation, it still is only able to solve relatively small instances and several areas offer opportunities for future research and development:

- **Different search heuristics:** Larger networks quickly increase the runtime. Future work could explore different search heuristics that guide the search better. For example, not all molecules are as likely to take part in a reaction, thus prioritizing certain molecules in the search might significantly improve the runtime.
- **Expand problem analysis:** Currently, the analysis step is still quite limited. An expansions could for example be to add more concentration profiles for each measured molecule (instead of the current single input-output example). If a experiment is done with different starting concentrations, dependencies between molecules could be found. Another expansion could be to analyse the current concentration profiles further, like the speed in which a concentration of a molecule increases or decreases.
- **Adding more constraints:** Since the search space still increases very fast when searching with a larger maximum depth. Incorporating more chemistry rules, such as looking at the energy available in the network might reduce the search space even further.
- **Attempting different simulators:** The current scoring function is based on a simulator that applies parameter tuning of the system of equations. However, this solver currently does not adhere to chemical properties, by incorporating a solver that is more physically accurate, the scoring of the networks might be more accurate.
- **Use bottom-up search:** While the staged pipeline works quite well, it is difficult to decide on stopping conditions for all the separate iterators. For example, if you move to the network synthesis stage before the previous stage has generated all required reactions, now the networks stage has to first exhaust all possible networks before you go back to the previous stage. Unifying the grammars in a single solver is arguably a more elegant solution, but does not have the required performance to solve the problems. Another promising attempt was made to switch from top-down to bottom-up search. However, during development of this thesis, the bottom up functionality unfortunately was still in development for the Herb.jl framework.
- **Expand the SMILES support:** The current molecules are a simplified version of SMILES. For example, atom isotopes and charge are not supported. As the solver improves, it might become feasible to support more detailed molecule descriptions. Adding this support might also create the opportunity for different constraints to be added making use of the introduced properties.

Bibliography

L. Cardelli, M. Češka, M. Fränzle, M. Kwiatkowska, L. Laurenti, N. Paoletti, and M. Whitby. Syntax-guided optimal synthesis for chemical reaction networks. In *Computer Aided Verification: 29th International Conference, CAV 2017, Heidelberg, Germany, July 24-28, 2017, Proceedings, Part II 30*, pages 375–395. Springer, 2017.

C. David and D. Kroening. Program synthesis: challenges and opportunities. *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences*, 375(2104):20150403, 2017.

M. Á. de Carvalho Servia, I. O. Sandoval, K. K. M. Hii, K. Hellgardt, D. Zhang, and E. A. del Rio Chanona. The automated discovery of kinetic rate models–methodological frameworks. *Digital Discovery*, 3(5):954–968, 2024.

E. Degrand, M. Hemery, and F. Fages. On chemical reaction network design by a nested evolution algorithm. In *Computational Methods in Systems Biology: 17th International Conference, CMSB 2019, Trieste, Italy, September 18–20, 2019, Proceedings 17*, pages 78–95. Springer, 2019.

S. Gulwani, O. Polozov, R. Singh, et al. Program synthesis. *Foundations and Trends® in Programming Languages*, 4(1-2):1–119, 2017.

B. Swinkels. Constraint propagation in program synthesis. Master's thesis, Delft University of Technology, Delft, The Netherlands, June 2024. Supervised by Dr. Sebastijan Dumančić and Tilman Hinnerichs.

J. P. Unsleber and M. Reiher. The exploration of chemical reaction networks. *Annual Review of Physical Chemistry*, 71(Volume 71, 2020):121–142, 2020. ISSN 1545-1593. doi: https://doi.org/10.1146/annurev-physchem-071119-040123. URL `https://www.annualreviews.org/content/journals/10.1146/annurev-physchem-071119-040123`.

M. Wen, E. W. C. Spotte-Smith, S. M. Blau, M. J. McDermott, A. S. Krishnapriyan, and K. A. Persson. Chemical reaction networks and opportunities for machine learning. *Nature Computational Science*, 3(1):12–24, 2023.

X. Zhou, L. Fu, P. Wang, L. Yang, X. Zhu, and C. G. Li. Drug-herb interactions between scutellaria baicalensis and pharmaceutical drugs: Insights from experimental studies, mechanistic actions to clinical applications. *Biomedicine & Pharmacotherapy*, 138:111445, 2021. ISSN 0753-3322. doi: https://doi.org/10.1016/j.biopha.2021.111445. URL https://www.sciencedirect.com/science/article/pii/S0753332221002304.

M. Ángel de Carvalho Servia, K. Kuok, Hii, K. Hellgardt, D. Zhang, and E. A. del Rio Chanona. Simplest mechanism builder algorithm (simba): An automated microkinetic model discovery tool, 2025. URL https://arxiv.org/abs/2410.21205.