Organic Chemistry Synthesis Problem as Artificial Intelligence Planning

by

Arman Masoumi

Bachelor of Science, Ryerson, 2012

A thesis

presented to Ryerson University

in partial fulfillment of the

requirements for the degree of

Master of Science

in the Program of

Computer Science

Toronto, Ontario, Canada, 2014

Organic Chemistry Synthesis Problem as Artificial Intelligence Planning

Master of Science 2014

Arman Masoumi

Computer Science

Ryerson University

Abstract

This thesis formulates organic chemistry synthesis problems as Artificial Intelligence planning problems and uses a combination of techniques developed in the field of planning to solve organic synthesis problems. To this end, a methodology for axiomatizing organic chemistry is developed, which includes axiomatizing molecules and functional groups, as well as two approaches for representing chemical reactions in a logical language amenable to reasoning. A novel algorithm for planning specific to organic chemistry is further developed, based on which a planner capable of identifying 75 functional groups and chemical classes is implemented with a knowledge base of 55 generic chemical reactions. The performance of the planner is empirically evaluated on two sets of benchmark problems and analytically compared with a number of competing algorithms.

Acknowledgements

I would like to express gratitude to my thesis advisor, Prof. Mikhail Soutchanski, who played a major role in completion of this thesis through his useful discussions and invaluable advices. Also, I am grateful to Megan Antoniazzi and Vitaliy Batusov whose help was essential for completion of this work. I am obligated to thank Megan for her assistance in experiments with state-of-the-art planners mentioned in this work. Also, Vitaliy's role was essential since we used his program to translate the reactions to the proper format for the state-of-the-art planners, and I wish to thank him.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Motivation and Problem Statement

Organic synthesis problem is one of the most important problems in organic chemistry concerned with constructing a target molecule from a set of existing molecules via chemical reactions. Some of the applications of solving organic synthesis problems, aside from education, include constructing new molecules with desired properties, degrading harmful molecules that do not degrade naturally, and drug discovery. Organic compounds and chemical reactions have been studied extensively by scientists around the world. As a result, a large number of chemical compounds and reactions have been identified which form the large knowledge bases for solving synthesis problems. Solving synthesis problems requires chemical expertise and is commonly done by trained chemists. However, synthesizing a relatively complicated molecule which takes several reactions to construct can be very challenging, even for experts in chemistry. The large number of known chemical reactions and molecules makes a synthesis problem intricate for chemists if the problem is slightly out of the norm and requires not-so-common reactions in order to be synthesized. In this thesis, we employ computers to the task of solving organic synthesis problems, by formulating organic synthesis as Artificial Intelligence (AI) planning problems.

In AI, planning is the problem of determining the actions that can transform an initial state to a goal state. Many variants of planning problems exist, ranging from those where actions have deterministic effects and everything about the environment is known, to those were actions have stochastic effects and partial knowledge about environment is provided. In all cases however, for a given domain, some actions are defined whose execution results in a new state by making changes in the state they were applied to. The computational task is to find the sequence of such actions whose effects transform an initial state to the goal state. In this thesis, the domain of the planning problems is organic chemistry. As such, in order to solve planning problems corresponding to organic synthesis problems, first a computer-based representation of chemical molecules and reactions, amenable to reasoning in a dynamic environment needs to be developed. Additionally, an efficient planning algorithm capable of solving planning problems in organic chemistry is required. Both requirements are addressed in this thesis.

1.2 Methodology

In order to represent and reason about chemical molecules and reactions, we develop a knowledge representation and reasoning (KRR) framework. KRR is a branch of AI which strives to encode knowledge in a logical language with the goal of facilitating useful reasoning. The challenge is in developing an expressive language for representing knowledge unambiguously while ensuring computational tractability of the procedures that derive implicit logical consequences. The KRR framework developed in this work for representing and reasoning about chemical reactions is based on the situation calculus (SC) [80]. SC is a well-known AI logical formalism for representation of and reasoning about dynamic domains. We introduce two approaches to representing chemical reactions in SC and discuss each approach's advantages and disadvantages. The developed framework enables representing and reasoning about generic chemical reactions and chains of reactions. Thus, using the framework it is possible to determine the effect of a reaction on a set of molecules, if the reaction is applicable.

We formulate organic synthesis problems as planning tasks in AI by appealing to the aforementioned framework for representing chemical molecules and reactions. The planning problems solved in this thesis are the following: given a set of initial molecules and a goal molecule (or a set of goal molecules), find the sequence of actions that transform the molecules in the initial state to those in the goal state. In order to solve the planning problems efficiently, we employ AI techniques and a combination of advancements made in planning community. A planning problem is first abstracted to convert the problem into a computationally easier problem. Second, heuristic search similar to those performed in state-of-the-art planners is conducted on the abstract problem to find abstract high-level solutions. Lastly, the abstract solutions are refined to concrete solutions for the organic chemistry synthesis problem.

1.3 Novelty and Significance

We are the first to represent chemical molecules and reactions is SC. Representing organic chemistry in SC differs from the common existing representations in that it is declarative. Furthermore, to the best of our knowledge, we are the first to formulate organic synthesis problems as planning tasks in AI, and as such our approach for solving organic synthesis problems differs from all previous related works. Moreover, we believe our algorithm solving the planning problems corresponding to synthesis problems is unique, although it is built on existing developments from the field of planning in AI.

Both the fields of organic chemistry and AI benefit from this work. Formulating organic synthesis as planning in AI makes it possible to apply a variety of techniques developed for efficient planning to solve organic synthesis problems, and as such, the field of organic chemistry benefits from this work. Additionally, organic chemistry, as a real-world domain with interesting complexities and features provides a valuable case study for planning in AI, and therefore the field of AI benefits from this work. Casting organic synthesis as planning and attempting to solve synthesis problems using techniques developed for planning sheds light on shortcomings and limitations of the techniques for planning, and as such provides valuable lessons that can be used to further nurture the field of planning.

The SC-based approaches developed in this work for representing chemical molecules and reactions assume a 2-dimensional environment and lack the ability to represent and reason about stereochemistry (3D organization of molecules and reactions). This forms the main conceptual limitation in our approach to representing molecules and reactions. As for the algorithm used for finding the sequence of reactions capable of synthesizing a goal molecule, two limitations can be mentioned. First, it relies on an external ontology of chemical molecules and functional groups in its procedure, and as such the limitations of the ontology (missing or incorrect knowledge) has negative effects on the algorithm. Second, the algorithm is not very efficient when there are multiple molecules in the initial state that have the same functional groups as the functional groups in the goal molecule. These limitations will be discussed more in the subsequent chapters of this thesis.

1.4 Contributions and Outline

The approaches developed in this thesis for representing and reasoning about chemical molecules and reactions, as well as some preliminary experimental results appeared in 3 peer-reviewed papers published in the proceedings of the international conferences [69, 70, 68]. This thesis subsumes the contributions made in the aforementioned papers and extends them. The list of contributions in this thesis is as follows:

1. We introduce a methodology to axiomatize organic chemistry, that is:
    - We show how arbitrary chemical molecules and functional groups can be represented as axioms in SC.
    - We discuss to which logical class the axioms representing molecules and functional groups in organic chemistry belong.
    - We show two approaches for axiomatizing chemical reactions in SC, namely the *micro* and *macro* approaches. In each approach:
        - successor state axioms (SSAs) are used to represent the effects (and non-effects) of the reactions compactly.
        - precondition axioms (PAs) are used to characterize the conditions that are needed for the reaction to be applicable.
        - For both approaches, we provide the templates for the axioms, discuss their properties, and present example axioms.
2. Building on techniques from the field of planning, we present a novel algorithm for solving organic synthesis problems.
3. Using this novel algorithm, we develop a planner called ChemPlanner, capable of solving organic synthesis problems, which:
    - has a knowledge base of 55 generic chemical reactions.
    - can identify 75 functional groups and chemical classes.
4. We compare the micro and macro approaches empirically when they are used for solving planning problems.

5. We perform experiments using ChemPlanner, and compare the results with a number of other competing planners, all of which are developed by us.

6. We compare the performance of ChemPlanner' algorithm with competing algorithms not developed by us.

    - We employ state-of-the-art AI planners to the task of solving organic synthesis problems and compare their performance with ChemPlanner.
    - We compare the performance of ChemPlanner with a competing algorithm on a set of benchmark problems from the literature. The competing algorithm in this case works in a different paradigm and does not formulate organic synthesis as AI planning.

The rest of this thesis is organized as follows. Chapter 2 introduces the background knowledge necessary for this research. Chapter 3 describes two approaches developed by us for representing chemical molecules and reactions in SC. Chapter 4 describes the planner developed in this work capable of solving organic synthesis problems. Chapter 5 presents and discusses experimental results obtained from ChemPlanner and compares them with some other algorithms. Chapter 6 ends the thesis with discussion about general conclusions, limitations and future work.

Chapter 2

Background

This chapter includes the background information that is used in the subsequent chapters of this thesis.

2.1 Organic Chemistry

Basic knowledge of organic chemistry is necessary for this research. This section provides background knowledge about organic chemistry. Readers familiar with organic chemistry can skip this section.

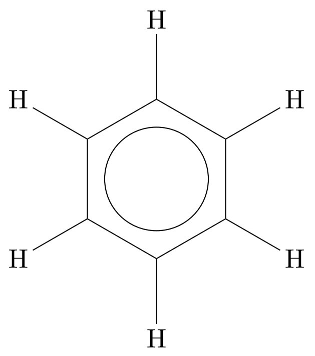
2.1.1 Atoms, Molecules, Functional Groups and Chemical Classes

Organic chemistry is the field devoted to study of organic material. In more details, this field explores the properties of molecules and structures that involve carbon atoms, and the reactions thereof. Other than carbon atom, some of the common atoms found in organic material are hydrogen, oxygen, and nitrogen. Atoms, as the building blocks of molecules have a major role in determining the molecule's characteristics and properties. Periodic table is a tabular representation of the atoms found so far, organized by their properties. On a high level, periodic table is constructed from periods (corresponding to rows in the table) and groups (corresponding to columns of the table). The atoms in a group generally tend to have similar properties which makes it useful to assign a name to a group of atoms. For the purpose of understanding this paper, it is sufficient to know the name of the two famous groups in the periodic table and what atoms they consist of. *Alkali metals* is the first group in the periodic table (the leftmost column) and *Halogens* are the atoms in the 17th group of the periodic table. Each element in the periodic table might have different *isotopes*. An isotope of an element is a variant of the element with the same number of protons, but different numbers of neutrons.

The atoms of the periodic table are capable of forming bonds with each other, to form molecules. Of course, not all atoms can form bonds with each other. In general, there are two kinds of chemical bonds: *covalent* and *ionic* bonds. In covalent bonds two atoms share electrons and that results in the bond between two atoms. If two atoms share a single electron, a *single bond* is formed between them. If they share two electrons, a *double bond* is formed between them, and *triple* and *quadruple* bonds are similarly formed by three and four shared electrons between two atoms, respectively. *Aromatic bonds* are formed when an electron is shared among more than two atoms. Aromatic bonds are common in rings of carbon, where a number of carbon atoms (typically 5 or 6) have bonds with each other. In ionic bonds however, electrons completely transfer from one atom to the other and that produces ions with negative and positive charges. Consequently, the ionic bond is due to attraction between two opposite electrically charged ions.

Moreover, there is a limit on how many bonds a certain atom can make, referred to as the chemical *valence* of the atom. For example, hydrogen, the atoms belonging to alkali metals and halogens can form only the single bond. Carbon atom C has valence of 4, meaning it can form 4 single bonds, or two double bonds, or a triple bond and a single bond, or a quadruple bond. Oxygen atom O has valence of 2, and valence number of sulfur atom S is 6 .

It is common to represent molecules as graphs, with atoms as vertices and bonds as edges of the graph. Since the focus of organic chemistry is molecules with carbon atoms, typically if a vertex is not explicitly named, it is assumed to be the carbon atom. Moreover, since hydrogen atoms are prevalent, sometimes they are not represented by chemists. In these cases, the atoms that are not saturated (i.e. have not formed all the bonds that they can) are implicitly assumed to have enough bonds with hydrogen atoms to make up the valence deficit. In our representation however, we usually explicitly display hydrogen atoms. A single edge between atoms represents a single bond and double edges (2 parallel lines) represents double (parallel) bonds. Triple and quadruple bonds are also similarly represented by increasing the number of parallel lines between the atoms. Aromatic bonds are usually represented with a straight line and a curved line. The most famous example of a molecule with aromatic bond is *benzene*, displayed below.

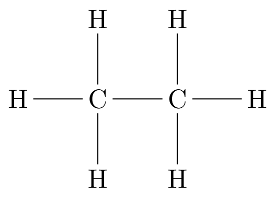


Benzene

Molecules are categorized into *chemical classes* based on their constitutive *functional groups*. Functional groups are specific groups of atoms within a molecule that are responsible for chemical characteristics of the molecule. The molecules that have the same functional groups behave similarly in chemical reactions. A chemical class contains numerous instances of specific molecules. Below, we introduce some of the common chemical classes and functional groups, and give examples of some instances of specific molecules belonging to each chemical class.

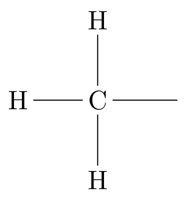
Alkane is the chemical class of molecules with the generic formula of C_nH_{2n+2}, where n is the number of carbons in the molecule. Alkanes are formed only from hydrogen and carbon atoms using only single bonds between carbon atoms. Alkanes do not have any cycles. Note that each carbon has four bonds (either with a carbon or a hydrogen) and each hydrogen atom is bound to a carbon.

Here is a famous example of an alkane:

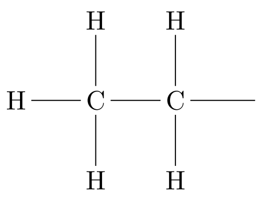

Ethane

Alkyls are the main functional groups of alkanes and are similar to alkanes in that the atoms constructing them are only carbon and hydrogen, and are only single-bonded. The only difference between alkanes and alkyls is that alkanes are saturated (every atom in the alkane uses its full capacity for forming bonds), while one carbon in an alkyls is not saturated with hydrogen atoms, and bridges to other functional groups. Alkyls have the generic formula of C_nH_{2n+1} and are usually represented with R, following the *Markush* syntax, which provides chemical symbols that are used to indicate a collection of chemicals with similar structures. Common alkyls and their names are described below.

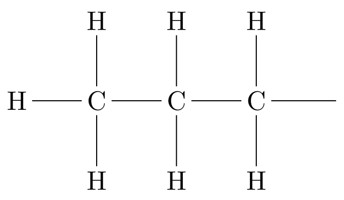
Alkyl with one carbon atom:

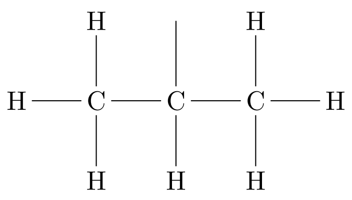

Methyl

Alkyl with two carbon atoms:

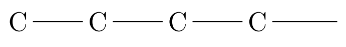

Ethyl

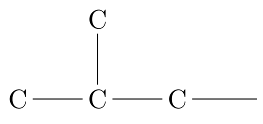
Alkyls with three carbon atoms (there are 2 variants -known as isomers- depending on whether a side chain or a middle chain is missing one hydrogen atom):

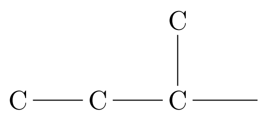

Propyl

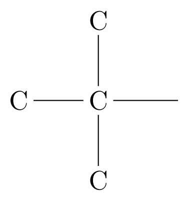

Isopropyl

To simplify the notation, the hydrogens of the alkyls with 4 or more carbons are not drawn. This does not lead to ambiguities because carbon has valence of 4 and the bond with missing hydrogen is always drawn explicitly. Below we reproduce all alkyl isomers with four carbon atoms:

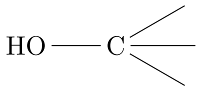

Butyl

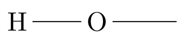

Isobutyl

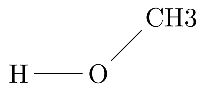

Sec-butyl

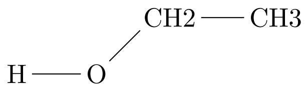

T-butyl

Another example of a chemical class is *alcohol*, with *hydroxyl* as the main functional group. Hydroxyl $-OH$ is the functional group where an oxygen atom has a single bond with a hydrogen atom, and from the other end forms a single bond with another atom. Alcohols are in general classes of molecules where a carbon atom has a bond with a hydroxyl, and three other single bonds to some other atoms. If the carbon atom has at most one bond with other carbon atoms, the alcohol is called a *primary alcohol*. If the carbon attached to the hydroxyl has two bonds with other carbon atoms, it constituents a *secondary alcohol*, and if it has three carbon atom neighbors, a *tertiary alcohol* is formed. Hydroxyl and generic alcohol are represented below, along with two famous alcohols:

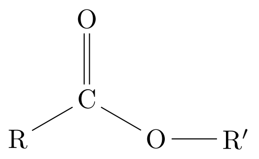

Alcohol


Hydroxyl

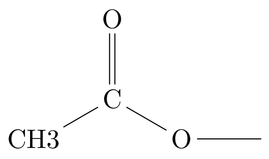

Methanol


Ethanol

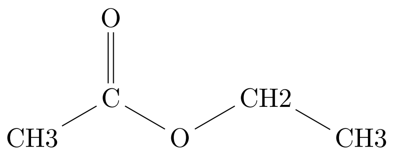
Ester is another chemical class, whose functional group is also called ester. Ester chemical class has the generic formula of $R - COO - R'$, where R and R' are alkyls. Ester functional group is $R - COO-$ with R being an alkyl. *Acetate* is the anion (molecules or atoms with negative electrical charge) of the form CH_3COO^- that can attach to some other molecules. If acetate attaches to an alkyl, it will form an acetate ester. This is an example in which a molecule can belong to more than one chemical class, as it has more than one functional group. In acetate esters, depending on the alkyl that the acetate anion attaches to, the name of the acetate ester varies. If it attaches to alkyl with one carbon atom (methyl), the ester is called *methyl acetate*, if it attaches to alkyls with two carbon atoms (ethyl), *ethyl acetate* is formed. Similarly, the names of the other acetate esters are determined by the alkyl that attaches to the acetate anion. You can visualize these chemical classes, functional groups and some specific molecules below:



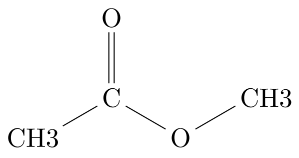
Ester



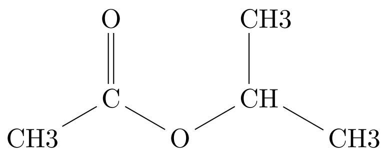
Acetate anion



Ethyl acetate



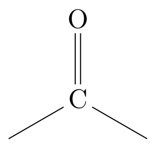
Methyl acetate



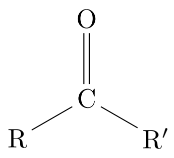
Isopropyl acetate

Carbonyl is a basic functional groups that appears in many other functional groups and chemical classes. It is characterized by a carbon atom double bonded to an oxygen atom, regardless of what other atoms the carbon is bonded to. *Ketone* is a chemical class that contains a carbonyl group, such that the carbon of the carbonyl has two single bonds with two carbon containing substituents. *Aldehyde* is another functional group, that is similar to ketone except that in aldehyde, one of the bonds with the carbon of the carbonyl is with a hydrogen atom.

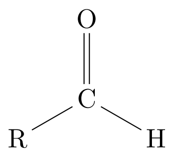
The structures of carbonyl, ketone and aldehyde are represented below, where R represents a carbon containing substituent.



Carbonyl

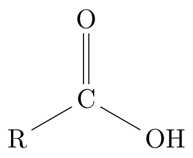


Ketone

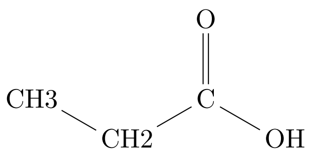


Aldehyde

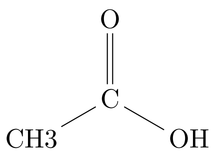
Carboxylic acids are structures with formula $R - COOH$, where R is a carbon substituent. Carboxylic acids are characterized by presence of at least one carboxyl group (of the form $-COOH$). Carboxylic acid chemical class, as well as some specific instances of this class are represented below.



Carboxylic acid

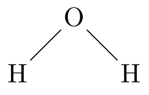


Propanoic acid

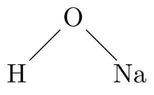


Acetic acid

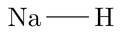
The term *acid* refers to molecules that donate protons (H^{+}) to other molecules (proton donors). A *base* on the other hand is defined as a proton acceptor [99]. *Water*, with the formula $H - O - H$ can take the role of an acid or a base, depending on the conditions. Acids and bases lack a general structure, but some of the famous acids and bases are introduced here. *Sodium hydroxide* and *sodium hydride* are among the strongest bases, and *hydrochloric acid* and *sulfuric acid* are among the strongest acids.



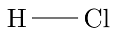
water



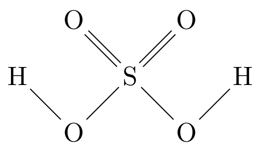
sodium hydroxide



sodium hydride



hydrochloric acid

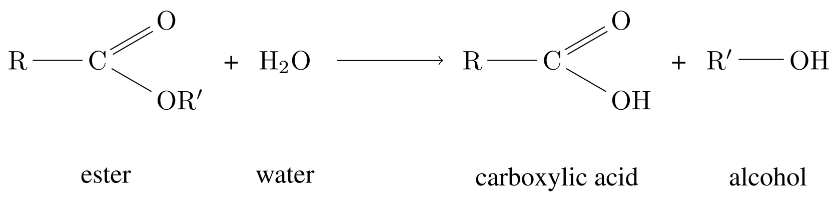


sulfuric acid

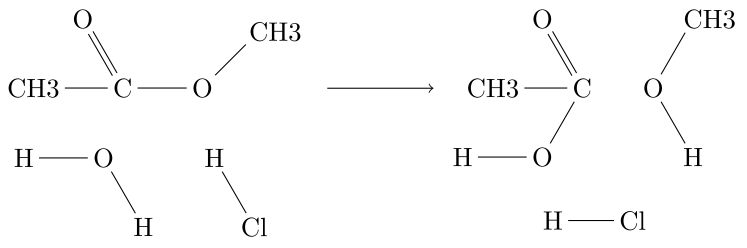
2.1.2 Chemical reactions

A chemical reaction is the process in which a set of molecules transform to another. The molecules before the occurrence of the chemical reaction are referred to as *reactants* (also known as *substrates* or *reagents*), and the new molecules produced are the *products* of the reaction. During a chemical reaction, some of the bonds between the atoms in the reactants break, and new bonds are formed to produce the products of the reaction. A chemical reaction usually is displayed with a chemical equation, where reactants of the reaction are displayed on the left hand side of an arrow (usually with "+" in between), and the product on the right hand side of the arrow. Chemical reactions sometimes need *catalysts* to occur. Catalysts are chemical compounds that are either necessary for, or speed up a chemical reaction, but they are not consumed and transformed to something else in the reaction. In other words, they appear unchanged both before and after a chemical reaction but affect the chemical reaction.

An example of a generic chemical reaction is *hydrolysis of esters*, in which an ester molecule reacts with water, in presence of a strong acid as the catalyst, to produce a carboxylic acid and an alcohol. The scheme for this reaction is displayed below, where the acid catalyst is not displayed to simplify the matters.



An instance of this reaction is the reaction between methyl acetate (as an ester) and water, with hydrochloric acid as the catalyst strong acid. This reaction produces methanol, as well as acetic acid.



A generic chemical reaction represents numerous instances of the chemical reaction. For example, in the above hydrolysis of esters reaction, different ester molecules can be used to produced different carboxylic acids and alcohols. Nevertheless, the essence of the reaction remains the same, as always the same bonds between atoms cleave and form. Generic chemical reactions describe the interactions between chemical classes and functional groups, as opposed to specific molecules. However, in organic chemistry, there are a few specific molecules that are used in chemical reactions commonly. These specific molecules are referred to as *chemical conditions*, or in short *conditions*. An example of a condition in chemical reactions is phosphorus tribromide PBr_3, which can react with alcohol molecules. Some other common conditions are hydrochloric acid HCl, lithium hydroxide LiOH and lithium aluminium hydride $LiAlH_4$, to name a few. The generic reaction between alcohols and phosphorus tribromide PBr_3 is displayed below:

$$\mathrm{R}\text{—}\mathrm{OH} + \mathrm{PBr3} \longrightarrow \mathrm{R}\text{—}\mathrm{Br} + \mathrm{PBr2OH}$$

2.2 Cheminformatics and Bioinformatics

Cheminformatics is the field of research that aims to use the computational power of computers to tackle problems in chemistry, with particular emphasis on storage and search of information relating to compounds. Years of research in this field has led to diverse models for representing chemical compounds and various methods for manipulating chemical structures. Bioinformatics, in a similar vein is the field of research focusing on applying computational techniques to storing, searching, and analyzing the biological data. Biological processes and pathways are made up of chemical reactions that occur in living organisms and are vital to their existence. As such, models developed in bioinformatics for representing biological processes and pathways are based on the same concepts as those developed in cheminformatics. In this section, we briefly review some of the related knowledge bases, as well as efforts done in these fields for representing molecules, functional groups, and reactions.

2.2.1 Existing Knowledge Bases and Limitations

Chemical Entities of Biological Interest (ChEBI) [27, 28] is a freely available database, dictionary and ontology of atoms, ions, functional groups, molecules and the like with focus on their relevance to biology. Each entity in ChEBI is distinguished with a unique ChEBI ID. Moreover, ChEBI provides an ontology classification incorporating only *isA* relations between molecular entities and a bit of mereology to show parthood relations between molecules and the constituent functional groups. However, the ontology does not fully represent the structure of the molecules. ChEBI is developed and maintained by chemist experts, and as such provides reliable information. However, although existing data are reliable, ChEBI suffers from some missing information, examples of which are missing digital representation of some of their entries, such as *hemiacetal*.

There are many (bio)chemical reaction and pathway databases developed, most of which are only commercially available and require paid licenses to be accessible. Nevertheless, they suffer from the following limitations:

- Not all reactions are generic.
- The full effects of the reactions are not represented and they are not balanced. In other words, in many cases only a subset of chemical bonds that are created or cleaved as the result of the reaction are displayed.
- The reactions are not atom mapped. That is, the one to one correspondence between atoms in the substrate and the atoms in the product of the reaction are missing.

The limitation regarding atom mapping of the reactions has a promising solution: there are existing software that are developed specifically for atom mapping of reactions, such as ChemAxons Standardizer [15].

ChemAxon's Standardizer accepts a variety of chemical formats and performs a variety of functions on it, including atom mapping and explicitly representing implicit hydrogens. However, addressing the other two limitations is more intricate, as there are no available software to solve the issues, and chemical expertise is required to provide the missing information.

2.2.2 Efforts for Modeling Molecules and Reactions

Computer-Oriented Representation of Organic Reactions [32], a book written by Shinsaku Fujita, describes different general ways of representing chemical compounds, as well as defines and describes some important concepts that are useful for representing organic reactions. Some of these concepts are *ITS*, *in-bond*, *out-bond*, *par-bond*, *reaction centers*, *reaction graphs* and *reaction strings*. ITS is a way of representing a chemical reaction in which three kind of bonds have been defined: out-bonds (bonds that exist only in the starting stage), in-bonds (bonds that exists only in the product stage), and par-bonds (bonds that exist in both starting and product compounds). Reaction centers are only the set of vertices that are active in the chemical reaction, meaning vertices that are incident to at least one in-bond or out-bond. In reaction centers, a vertex could represent an atom, a compound (a set of atoms, depending of the level of descriptiveness) or by a ball (unnamed vertex). The latter describes the Reaction Graphs (RG). From this definition derives that reaction centers are more specific to a chemical reaction, but reaction graphs are more general and can describe a set of reactions. ITS is in spirit similar to "superimposed reaction graphs", a formalism for modeling chemical reactions developed by G.E. Vladutz [96, 95]. In superimposed reaction graphs, the atoms that do not participate in bond alterations (the atoms with no adjacent in-bonds or out-bonds) are omitted to produce a sub-graph called "superimposed reaction skeleton graph" which focuses on the alterations that a chemical reaction make. In the same work, Fujita also discusses the various reaction graphs such as "odd-membered cyclic reaction graphs", "even-membered cyclic reaction graphs" and a few other reaction graphs. The idea of "imaginary rings" is also introduced and discussed. The concept of imaginary rings is nothing new when we are familiar with ITS. In fact any ring structure appearing in ITS is referred to as imaginary ring or ITS ring. Additionally, linear coding of reaction types and enumeration of reaction graphs and reaction center graphs are discussed by Fujita. Lastly, a short discussion about synthesis pathways is considered. A synthesis pathway involves two or more unit reactions.

Rossello and Valiente [81] introduce a model for representing chemical molecules and chemical reactions, based on considering chemical molecules as graphs and chemical reactions as a structure that contains the substrate and product graphs. The authors describe chemical reactions in [82] by edge relabeling graph transformation rules. They also introduce the PerlMol toolkit, an object-oriented module written in Perl which provides objects and methods for representing molecules, atoms, and bonds. The toolkit is also capable of doing substructure matching and reading and writing files in various formats. In this work, molecules are represented with an abstract notion of *mol* which provides a match method that finds subgraph isomorphisms. Reactions are created using a constructor that takes a substrate and product *mol* object in addition to a map of substrate atoms to product atoms. As such, only specific reactions are representable in this work.

Talcott [87] presents Pathway Logic, a symbolic biology approach for modeling biological processes and uses it to model signal transduction processes. Symbolic systems biology aims to study biological processes both qualitatively and quantitatively as integrated systems. Symbolic and logical models allow incomplete knowledge to be represented with different levels of abstractions. Pathway Logic, is based on *rewriting logic* and is developed in *Maude*. Rewriting logic is a logical formalism consisting of *states* and *rules*. States represent states of the system using an algebraic data type and each rewrite rule represents a biological process. Collection of rules together with the underlying type specification forms the Pathway Logic knowledge base. Briefly, signal transduction happens when a cell's receptor transmits a signals to the appropriate component in the cell informing it about the cell's environment. The Pathway Logic knowledge base can be analyzed and executed using Maude, which is a language and a tool based on rewriting logic.

In [55], Dumontier describes chemical knowledge representation using the Web Ontology Language (OWL) where there are objects such as Molecules, Atoms and Rings. In this work, Molecules or Rings are related to Atoms by 'hasProperPart', Molecules and Rings are related to each other by 'hasPart', and Atoms are connected to each other via symmetric 'hasBondWith' relations. Later, Chepelev and Dumontier [17] use semantic web technologies to represent polyatomic chemical entities, their substructures, bonds, atoms, as well as reactions. They use ChEBI IDs for describing molecules and utilize relations such as 'hasProperPart' and 'isPartOf'. As for chemical reactions, they utilize relations like 'hasInput', 'hasOutput', 'agent' with links to ChEBI IDs. In essence, they are not concerned about internal mechanism of chemical reactions, rather consider a reaction as ID_1 'transforming into' ID_2. In a related work [18], a formal framework based on Semantic Web technologies for automatic design of chemical ontology is presented. They introduce the terminology for structures that contain other structures (but are not contained) as *consensus chemical features* and base their work on this to introduce children of a chemical class. The work of Hastings et al. in [40] is very similar in spirit to the other works discussed in this paragraph.

Magka et al in [67] switch to logic programming for representing and reasoning about chemical molecules to benefit from negation as failure. They introduce *description graphs* (DGs) which uses relations like 'hasAtom' and 'bond' and translate DGs to logic programs with function symbols. Kutz et al [57] introduce some examples of highly symmetric molecules, constituted almost entirely by carbon atoms. They claim such molecules must be characterized by their shape or topology and that it is not possible to reason at the class level about highly symmetric molecules using DL-safe rules. Furthermore, they argue that using the work of Magka (DGLP framework), it is not possible to express the properties of these complex molecules as a whole, as first order logic is not expressive enough and second order logic is required.

There has also been several line notations developed for representing molecules. SLN [5] is a line notation with the standard nomenclature developed by International Union of Pure and Applied Chemistry (IUPAC) for representing atoms. In this representation, attributes of atoms are expressed in square brackets following the atom name. Hydrogen atoms are explicitly represented, for example the SLN for methane is CH4. Wild card atoms are also supported in SLN which include Markush syntax for R groups. Another line notation is the IUPAC International Chemical Identifier (InChI) [42]. InChI has a layered structure for modularizing the representation and is not designed for substructure searching. Another well-known line notation is SMILES [1] which, like InChI, allows a canonical serialization of molecular structure. While InChI is an open source project, SMILES is proprietary and different generation algorithms produce different SMILES of the same compound. Unlike InChI, which uses a standard canonicalization algorithm, SMILES canonicalization varies by software package. Similar to InchI, SMILES also supports stereochemistry (3 dimensional representation of molecules). SMARTS [1] is an extension of SMILES that allows the user to describe a (sub)molecule, and as such is predominantly used for substructure searching in molecules. Additionally, SMIRKS [1] is a language designed for representing generic chemical reactions which has SMILES and SMARTS as its foundation.

Another line notation, the Molecular Query Language (MQL) [79] was designed to allow more complex, problem-specific search methods in cheminformatics. In contrast to the widely used SMARTS queries, MQL provides for the specification of spatial and physicochemical properties of atoms and bonds. Additionally, it can easily be extended to handle non-atom-based graphs, also known as *reduced feature* graphs. The query language is based on an extended Backus-Naur form (EBNF) using JavaCC. Lastly, there is Wisswesser Line Notation (WLN) [97]. WLN provided a way to describe a molecule, but it did not produce a canonical name. That is, it did not include a set of rules which could be applied to a molecule to get the same name every time. It, like the IUPAC nomenclature, required a chemist to identify the parent, and chemists have different opinions on how that is done. WLN is relatively complex which makes it intractable for many classes of compounds. However, as the advantage, WLN is remarkably compact, especially when compared to SMILES and InChI and functional group recognition is easy in this representation. WLN's complexity prevented widespread adoption as encoding rules for correct specification of WLN into a computer proved difficult.

As for related work on functional groups, [84] provides a conceptual model for defining and detecting functional groups. In this model, a primary functional group is defined as a group center (Gc), acting as a backbone, bonded to terminal atoms (Ta) and skeletal carbon atoms (Cs). As such, functional groups can be categorized by the number of atoms in their group centers. Moreover, combinations of primary functional groups form functional group assemblies. The notion of functional groups is also extended to include characteristic groups containing only carbon atoms, as these groups are common and play an important role in reactivity of the molecules. Based on this conceptual model, an ontology of functional groups is also developed, called *FOnt*. The authors of [88] design a reactivity database for functional groups as follows. They analyze the reaction database and extract data regarding which functional groups seem to have no effect on each other (inert data), which functional groups cause change or destruction of another (interfering data), as well as in presence of competing functional groups, which functional group is more likely to change (relevant reactivity or relative rate data). Haider [38] presents a utility capable of recognizing and identifying 128 functional groups in MolFile format [23]. MolFile is a popular format for representing molecules, based on connection table representation of the molecules. Haider uses the idea of *key atoms* to characterize the functional groups, a concept we borrow in our representation. Lastly, [56] reports 489 functional groups of biological interest that have been identified in National Cancer Institute (NCI) and KEGG databases, and are accessible from a database named Biochemical Substructure Search Catalogue (BiSSCat).

2.3 The Situation Calculus

In this section, the Situation Calculus (SC) is introduced, as the underlying formalism used for representing chemical molecules and reactions in this work. In what follows, the symbols $\wedge$, $\vee$ and $\neg$ are used to represent the boolean connectives "and", "or" and "not" respectively. Additionally, symbol $\rightarrow$ represents implication (if...then...) and symbol $\leftrightarrow$ represents equivalence (if and only if). Also $\forall$ stands for the universal quantifier and $\exists$ stands for the existential quantifier. Finally, $\cup$ represents set union. The symbols that start with capital letters are constants and those starting with lower case letters are variables. An exception is the function symbols that start with lower case letters but are constants. In axioms, all free variables are implicitly universally ($\forall$) quantified at front.

The *situation calculus* [60, 80, 13] is a logic formalism designed for representing and reasoning about dynamical domains. In recent years, it has been considerably extended beyond the original language to include stochastic actions, concurrency, continuous time and so on [80]. Basic ingredients of SC consist of *actions* a, *situations* s and *fluents*.

Actions are used to represent changes occurring in a domain. Formally, Actions, $A(\vec{x})$, where $\vec{x}$ is a tuple of distinct *object* variables, are first order logic (FOL) terms consisting of an action function symbol A and its arguments $\vec{x}$. The variables can be instantiated with objects of the domain, in which case a grounded action is produced. For example, the action representing a man picking up a box can be denoted by $pickup(Man_1, Box_1)$. In this example Man_1 and Box_1 are some of the objects of the domain. There are usually a finite number of objects in a domain. Similarly, $drop(Man_1, Box_1)$ represents the action of Man_1 dropping the Box_1, and $paint(Box_1, Red)$ represents the action of painting Box_1 to the colour Red.

Performing actions will result in changing the *situation*. A situation is a first-order term denoting a sequence of actions. The special constant S_0 denotes the initial situation, namely the empty action sequence. The function $do(a, s)$ denotes the new situation that results from performing action a in situation s. For example, the situation where Man_1 picks up Box_1 and then Man_2 picks up Box_2 can be represented by
$do(pickup(Man_2, Box_2), do(pickup(Man_1, Box_1), S_0))$.
A situation s is called a proper sub-history of another situation s', denoted by $s \prec s'$ if s' contains the sequence of actions corresponding to s but is not equal to it. $s \preceq s'$ is the abbreviation for $s = s' \vee s \prec s'$.

Fluents $F(\vec{x}, s)$ are predicates whose values may vary from situation to situation, and therefore are predicates with the last argument s being a situation. They generally describe those features of the application domain that may change when actions are executed. As an example, consider the fluent $OnTheGround(Box_1, s)$ with value true if in situation s the box Box_1 is on the ground. Another example could be $Holding(Man_1, Box_1, s)$ which its value is true if in situation s, Man_1 is holding the Box_1 in his hands.

A basic action theory (BAT) D is a set of axioms in the situation calculus that is used to model actions, their preconditions, their direct effects and initial values of the fluents. The BAT is defined as the union of five groups of axioms:

$D = \Sigma \cup D_{ap} \cup D_{ss} \cup D_{una} \cup D_{S_0}$ where

1. Σ is the foundational axioms for situations. It consists of the following four axioms:

   - $\neg s \prec S_0$ meaning there is no situation that is a proper sub-history of S_0.
   - $s \prec do(a, s') \leftrightarrow s \preceq s'$ meaning any situation s is a proper history of $do(a, s')$ if and only if s is a history of s'.
   - $do(a_1, s_1) = do(a_2, s_2) \leftrightarrow a_1 = a_2 \wedge s_1 = s_2$ indicating the unique name axiom for situations.
   - $(\forall P).P(S_0) \wedge (\forall a, s)[P(s) \leftrightarrow P(do(a, s))] \leftrightarrow (\forall s)P(s)$ which is the second-order induction axiom on situations.

   The foundational axioms guarantee that the situations in any model can be represented as a tree.

2. D_{una} denotes the unique name axioms for actions, meaning two different action names represent two different actions.

3. D_{S_0} denotes initial theory, meaning the set of axioms representing the initial state of the environment, before any action has occurred. Typically, this is a set of FOL formulas constructed from fluents with S_0 as the only situation argument. In initial theory we might have sentences with no situation argument in them, for example $Supervisor(Jack)$ or $Supervisor(x) \rightarrow Human(x)$. Initial theory may include external static ontologies or facts that do not change. Also, notice that initial theory might be incomplete if not all the facts about the initial state of the environment is known.

4. D_{ap} is a set of action precondition axioms. These axioms characterize possibility of executing an action. Precondition axioms (PAs) use the distinguished predicate $Poss(A(\vec{x}), s)$ which is true if $A(\vec{x})$ is possible in situation s. There is one axiom for each action term $A(\vec{x})$, with syntax $Poss(A(\vec{x}), s) \leftrightarrow \Pi_A(\vec{x}, s)$. $\Pi_A(\vec{x}, s)$ represents the preconditions of action A: A is possible if and only if (use the bi-conditional $\leftrightarrow$ for iff) the logical condition $\Pi_A(\vec{x}, s)$ holds in s. In the example that we have been following, a possible precondition axiom could be:

   $$Poss(pickup(man, box), s) \leftrightarrow \neg Holding(man, x, s) \wedge OnTheGround(box, s).$$

   This axiom means it is possible for a man to pickup a box if the man is not holding any objects in his hands and the box is on the ground in the situation s. Another example is that of $drop$:

   $$Poss(drop(man, box), s) \leftrightarrow Holding(man, box, s).$$

   This axiom is stating that it is possible for man to drop a box, if he is holding the box in situation s.

5. D_{ss} is a set of axioms characterizing the effects of the actions on the fluents. For example, the effect of picking up a box is that the box will not be on the ground anymore and the agent who picked it up will be holding the box. A certain action might change the truth value of a certain fluent to false and might change the value of another fluent to true. But not all actions will affect all fluents, for example when a man picks up a box, this action will not affect the color of the box. Thus, $Colour(Box_1, Blue, s)$ will not change as a result of $pickup$ action. Similarly, the action $paint(box, colour)$ does not affect the truth value of $Holding$, but affects $Colour$ fluent. D_{ss} is the set of *successor state axioms* (SSAs), that are used to represent the effects of the actions on the fluents. The idea behind successor state axioms is that a fluent becomes true after executing an action if the action causes it to become true, or the fluent remains true if it was already true and the action taken did not cause it to become false. Otherwise, the fluent becomes false, if the most recently executed action has a negative effect on the fluent.

   More formally, there is one axiom for each fluent $F(\vec{x}, s)$, with syntactic form $F(\vec{x}, do(a, s)) \leftrightarrow \Phi_F(\vec{x}, a, s)$, where $\Phi_F(\vec{x}, a, s)$ is a formula *uniform* in s with free variables among $a, s, \vec{x}$. A situation calculus formula $\psi(s)$ is uniform in s, if s is the only situation term mentioned in $\psi(s)$, the formula ψ has no occurrences of the predicates $Poss, \prec$, and has no quantifiers over situations. Each SSA has the following generic form:

$$F(\vec{x}, do(a,s)) \leftrightarrow \bigvee_i a = \mathit{PosAction}_i(\vec{x}) \wedge \gamma_i^+(\vec{x}, s) \vee$$
$$F(\vec{x}, s) \wedge \neg\big(\bigvee_j a = \mathit{NegAction}_j(\vec{x}) \wedge \gamma_j^-(\vec{x}, s) \big),$$

where $\mathit{PosAction}_i$ is an action that makes the fluent F true and $\gamma_i^+(\vec{x}, s)$ is the formula expressing a context in which this positive effect can occur; similarly, $\mathit{NegAction}_j$ is an action that can make the fluent F false if the context formula $\gamma_j^-(\vec{x}, s)$ holds in s. If the executed action a is none of these, then the truth value of F remains unchanged (a has no effect). SSAs characterize the truth values of the fluent F in the next situation $do(a, s)$ in terms of fluents in the situation s and they represent non-effects of actions compactly (because of implicit universal quantifier $\forall$ over action variable a). For example, the successor state axiom for *Holding* fluent is as follows:

$$Holding(man, box, do(a, s)) \leftrightarrow$$
$$a = pickup(man, box) \vee$$
$$(Holding(man, box, s) \wedge a \neq drop(man, box)).$$

This SSA is stating that man is holding box if the last action executed was the man picking up the box, or the man already was holding the box, and the last action was not to drop it.

BATs might also be augmented with *abbreviations*, also known as *derived predicates* [90]. Abbreviations look similar to fluents in the sense that they are also predicates with their last argument a situation. Similar to fluents, their truth value can vary from situation to situation. However, abbreviations differ from fluents in that the actions do not directly affect them. Instead, the effects of the actions on abbreviations are implicit. So, there are no SSAs for abbreviations. Syntactically, axioms defining abbreviations are formulas *uniform* in s. In the SC, the axioms defining abbreviations using formulas uniform in s are called *state constraints*. As an example, consider the abbreviation $HasWorkToDo(man, s)$, which is true if man is not holding anything in situation s, and there is a blue box on the ground.

$$HasWorkToDo(man, s) \overset{def}{=}$$
$$\neg\exists box(Holding(man, box, s)) \wedge \exists box(Colour(box, Blue, s) \wedge OnTheGround(box, s)).$$

Notice that there is no action that can make the predicate $HasWorkToDo$ true directly. This is why the most intuitive way to have it properly defined is through a state constraint. Since actions have direct effect on the fluents $Holding(man, box, s)$, $Colour(box, colour, s)$ and $OnTheGround(box, s)$, they also have indirect effect on the predicate $HasWorkToDo$.

Sometimes, abbreviations can be eliminated, but it is convenient to keep them to make other axioms more succinct. In addition, axioms defining abbreviations help to make a logical theory more modular. For example, abbreviations can occur in the precondition axioms and can represent common terms of an application domain. For example, consider the hypothetical action $confirmWorkIsDone(supervisor)$, which is possible if $supervisor$ is a supervisor, and there is no man that has work to do. The PA for $confirmWorkIsDone(supervisor)$ is as follows:

$$Poss(confirmWorkIsDone(supervisor), s) \leftrightarrow$$
$$Supervisor(supervisor) \wedge \neg\exists man(HasWorkToDo(man, s)).$$

In this PA, the abbreviation $HasWorkToDo$ is used on the right hand side of the PA, and makes it easier to comprehend the PA, as well as improves modularity and succinctness.

In this thesis, we will argue that the set of abbreviations satisfy certain syntactic restrictions similar to recursive stratified rules in Datalog [3], introduced to consider recursive queries over databases. This is reviewed in the next section.

2.4 Logical Background

In this section, some logical background is provided that are referred to in the subsequent chapters and sections. It is assumed that the reader is familiar with first order logic (FOL), and as such the basic terminology of FOL is not introduced. The logical background includes a brief introduction to Datalog and stratified Datalog$^{\neg}$, SLDNF resolution and Lloyd-Topor transformation.

2.4.1 Datalog

Datalog [3] is a database query language influenced by logic programming with elegant syntax for expressing recursion, which has influenced development of other popular query languages. A Datalog *program* is a finite set of Datalog *rules*, where each rule is composed of two parts of *head* and *body* with the following syntactic form:
$head \leftarrow body.$
The head of the rule is an atom (in the context of FOL) with the syntactic form of $R(u)$, where R is a relation name, and u is a tuple. The body is either empty, or syntactically defined as
$R_1(u_1), \ldots, R_n(u_n),$
where $R_1, \ldots, R_n$ are relation names and $u_1, \ldots, u_n$ are tuples. For a Datalog rule to be valid, it has to be *range restricted*, meaning each variable occurring in the head tuple (u) must at least appear once in one of the tuples of the body $(u_1, \ldots, u_n)$. It is important to note that neither negation, nor function symbols are allowed in Datalog rules. In other terms, a Datalog rule is a function-free Horn clause. As usual in Horn clauses, a rule with empty body is called a *fact*. Also, any $R_1(u_1), \ldots, R_n(u_n)$ either forms a rule body, or a *query*. A relation (also known as predicate) that only appears in the body of the rules in a Datalog program, is referred to as extensional database (EDB). On the contrary, a relation that appears in the head of some rule in the program is termed intentional database (IDB). The values for EDB predicates are given via a *database instance*, also known as *input database*. The values of the IDB predicates are computed by the program. Consider a famous Datalog program below.

$$T(x, y) \leftarrow G(x, y) \tag{2.1}$$

$$T(x, y) \leftarrow G(x, z), T(z, y). \tag{2.2}$$

The above example is the Datalog program that computes the transitive closure of a graph, when the edges of the graph are given by the binary relation G. In the above example, the relation T is IDB, and G is EDB.

Formally, there are three equivalent but different ways of defining semantics for Datalog programs, namely *model theoretic*, *proof-theoretic* and *fixpoint* approaches. A brief description of each is presented next.

In the model theoretic approach, each rule in a Datalog program is associated with a corresponding first order sentence. The semantics in this approach are defined as the minimal model satisfying the first order sentences, and

the set of facts in the input database. It is proved that for each Datalog program and input there is always a unique minimal model. The correspondence between Datalog rules and first order sentences is done by associating each rule $R_1(u) \leftarrow R_2(u_2), \dots, R_n(u_n)$ with the following sentence:
$\forall x_1, \dots x_m(R_2(u_2) \wedge \dots \wedge R_n(u_n) \rightarrow R_1(u))$,
where $x_1, \dots, x_m$ are all the variables in the Datalog rule. For example, the corresponding sentences to the rules defining transitive closure are
$\forall x, y(G(x, y) \rightarrow T(x, y))$
$\forall x, y, z(G(x, z) \wedge T(z, y) \rightarrow (T(x, y)))$.

The proof-theoretic approach is based on the idea that given a Datalog program, the answer to a query consists of the set of facts that can be proven from the EDB facts using the rules of the program. In general, there are two ways of deriving new facts: *bottom-up* evaluation and *top-down* evaluation. In the bottom-up approach, the starting point is the known facts, and the rules are applied to derive all the new facts. It is not a goal directed approach, and many unrelated facts might be derived before deriving a fact of desire that can help answer the query. The top-down approach however, starts with query as the goal with a goal reduction mindset. Starting from the query, it derives intermediate goals using the facts and the rules of the program, and continues this approach recursively until all intermediate goals are proven, in which case the query succeeds (possibly with a proper substitution for the variables), or cannot be proven, in which case the query fails. This is the main idea behind the technique called *resolution*. A famous top-down resolution algorithm is called *SLD*.

In the third approach, fixpoint, the semantics are defined as a particular solution of a fixpoint equation. This approach is closely related to the bottom-up evaluation, and is based on the idea that one can use the rules of the Datalog program to define the *immediate consequence operator*. The operator is applied to the facts of the database to derive new facts, until no further facts can be derived, at which point a fixpoint is reached.

2.4.2 Stratified Datalog$^\neg$

Datalog$^\neg$ extends Datalog with allowing negation $\neg$. Similar to Datalog programs, a Datalog$^\neg$ program is a finite set of rules composed of head and body. Syntactically, there are two differences between a Datalog and Datalog$^\neg$ program. First, the atoms in the body of a Datalog$^\neg$ can be negated. Second, the equality predicate is introduced in Datalog$^\neg$, and $\neg = (x, y)$ is denoted with $x \neq y$.

Semantically however, introduction of negation prevents natural extension of the semantics discussed above to Datalog$^\neg$. For example, in the model-theoretic approach, the guarantee of a unique minimal model is lost in Datalog$^\neg$ programs. To counter this problem, a syntactic restriction called *stratification* is applied on the rules of Datalog$^\neg$, which causes the restricted class to extend well with the semantics introduced above.

Semi-formally, a stratification of a Datalog$^\neg$ program P is an ordered partition $P_1, \dots, P_n$ of the program such that

1. all rules defining an IDB predicate R appear in the same partition.
2. if a non-negated IDB predicate R appears in the body of a rule with head containing the predicate R', then the rules defining R should be in a partition before those defining R' or the same partition in which R' is defined.

3. if a negated IDB predicate R appears in the body of a rule with head containing the predicate R', then the rules defining R should be in a partition strictly before those defining R'.

Given a stratification $P_1, \dots, P_n$ for a program, each P_i is called a *stratum*, and the program is called *stratified* or *stratifiable*.

2.4.3 SLDNF Resolution

A famous deduction machinery, practicing the top-down proof theoretic approach is the *SLDNF* resolution. SLDNF is employed by usual Prolog systems. In SLDNF, the evaluation of negated literals is done by *negation as failure*. The idea behind negation as failure is to derive $\neg A$ if A cannot be proved. It is the manifestation of the *closed world assumption*: A statement that cannot be proved true is assumed false.

On a high level, SLDNF works as follows. Given a program and a query $g_1, \dots, g_n$, the procedure evaluates each sub-goal g_i in the query in order from left to right. If a sub-goal is negated, the non-negated form of the sub-goal is considered first, and the original sub-goal succeeds only if the non-negated sub-goal cannot be proven. The evaluation of positive sub-goals is done by considering the rule heads of the program in order from top to bottom for a match with the sub-goal's predicate. In case of a match, if the rules' body is empty, the proper substitution for the variables in the sub-goal is derived, or if a substitution is not found, the subsequent rules of the program are considered, if any. If the rule has a body, the body of the rule replaces the sub-goal, reducing the sub-goal to the body of the rule defining it. At each stage when a substitution is made, a choice point is created. If any sub-goal fails in the course of execution, all variable bindings that were made since the most recent choice-point are undone, and execution continues with the next alternative of that choice-point. This procedure is continued until all sub-goals are proven, or cannot be proven.

However, a problem in pure top-down approaches, including SLD and SLDNF is that they are not guaranteed to terminate on a recursive program, even if the program is stratified. Consider the example program computing transitive closure once again, and the below relation G defining the edges of the graph in the database instance:
$G = \{(a, b), (b, a), (c, a)\}$
It is clear that (a, c) is not in the transitive closure of graph G. However, the query $T(a, c)$ fails to terminate on this input using the SLDNF algorithm. To elaborate, we trace the steps of SLDNF when evaluating the query $T(a, c)$:

1. The intermediate goal $G(a, c)$ is formed, from rule 2.1.
2. The intermediate goal $G(a, c)$ fails, since it cannot be matched against the input database.
3. The intermediate goal $G(a, z), T(z, c)$ is derived next, from rule 2.2.
4. $G(a, z)$ succeeds with $z = b$, and the next intermediate goal $T(b, c)$ is formed.
5. The intermediate goal $G(b, c)$ is formed, from rule 2.1.
6. The intermediate goal $G(b, c)$ fails, since it cannot be matched against the input database.
7. The intermediate goal $G(b, z), T(z, c)$ is formed, from rule 2.2.
8. $G(b, z)$ succeeds with $z = a$ and the next intermediate goal $T(a, c)$ is formed.

Notice that the intermediate goal derived in step 8 is the same as the query we started from. As such, an infinite loop is formed which will never terminate following the SLDNF resolution.

In pure top-down approaches, the naive attempt to prove each goal independently is the main cause for this non-terminating problem. This behaviour can be avoided by memorizing which intermediate goals have been formed and whether they have been tried before. Of course, this will result in a mixed strategy of top-down and bottom-up approaches, although the top-down approach is still dominant. This technique of remembering the intermediate results and goals is called *memoization* or *tabling*, and is employed in some Prolog implementations, including *YAP* [25] and *XSB* [86].

2.4.4 Lloyd-Topor Transformation

Lloyd-Topor transformation is a method that systematically translates extended logic programs with logical connectives and explicit quantifiers into normal clausal form, suitable to be represented in logic programming languages such as Datalog. More specifically, the transformation is given by a number of operations that can be applied to clauses. The rules are then applied again and again wherever possible until no rule applies. The Lloyd-Topor transformation rules are as follows, where W, W_1, W_2, ψ and ϕ are arbitrary first order formulas, and ϕ and ψ might be missing:

(a) Replace $\psi \wedge \neg(W_1 \wedge W_2) \wedge \phi \rightarrow A$ with $\psi \wedge (\neg W_1 \vee \neg W_2) \wedge \phi \rightarrow A$.

(b) Replace $\psi \wedge (W_1 \vee W_2) \wedge \phi \rightarrow A$ with $\psi \wedge W_1 \wedge \phi \rightarrow A$ and $\psi \wedge W_2 \wedge \phi \rightarrow A$.

(c) Replace $\psi \wedge \neg(W_1 \vee W_2) \wedge \phi \rightarrow A$ with $\psi \wedge \neg W_1 \wedge \neg W_2 \wedge \phi \rightarrow A$.

(d) Replace $\psi \wedge (W_1 \rightarrow W_2) \wedge \phi \rightarrow A$ with $\psi \wedge (\neg W_1 \vee W_2) \wedge \phi \rightarrow A$.

(e) Replace $\psi \wedge \neg(W_1 \rightarrow W_2) \wedge \phi \rightarrow A$ with $\psi \wedge \neg(\neg W_1 \vee W_2) \wedge \phi \rightarrow A$.

(f) Replace $\psi \wedge (W_1 \leftrightarrow W_2) \wedge \phi \rightarrow A$ with $\psi \wedge (W_1 \rightarrow W_2) \wedge (W_2 \rightarrow W_1) \wedge \phi \rightarrow A$.

(g) Replace $\psi \wedge \neg(W_1 \leftrightarrow W_2) \wedge \phi \rightarrow A$ with $\psi \wedge \neg((W_1 \rightarrow W_2) \wedge (W_2 \rightarrow W_1)) \wedge \phi \rightarrow A$.

(h) Replace $\psi \wedge \neg\neg W \wedge \phi \rightarrow A$ with $\psi \wedge W \wedge \phi \rightarrow A$.

(i) Replace $\psi \wedge (\forall x_1, \ldots, x_n)W \wedge \phi \rightarrow A$ with $\psi \wedge \neg(\exists x_1, \ldots, x_n)\neg W \wedge \phi \rightarrow A$.

(j) Replace $\psi \wedge \neg(\forall x_1, \ldots, x_n)W \wedge \phi \rightarrow A$ with $\psi \wedge (\exists x_1, \ldots, x_n)\neg W \wedge \phi \rightarrow A$.

(k) Replace $\psi \wedge (\exists x_1, \ldots, x_n)W \wedge \phi \rightarrow A$ with $\psi \wedge W \wedge \phi \rightarrow A$.

(l) Replace $\psi \wedge \neg(\exists x_1, \ldots, x_n)W \wedge \phi \rightarrow A$ with $\psi \wedge \neg p(y_1, \ldots, y_k) \wedge \phi \rightarrow A$, where $y_, \ldots, y_k$ are all the free variables in $(\exists x1, ..., xn)W$, and p is a new predicate symbol not already appearing in the formulas.

For example, consider the first order formula below, expressing the sufficient condition to conclude $subset$ relation:

$\forall x, y, z((member(z, x) \rightarrow member(z, y)) \rightarrow subset(x, y))$.
The Lloyd-Topor normal form of the formula above, after having applied the Lloyd-Topor transformation is:
$\neg p(x, y) \rightarrow subset(x, y)$,
$member(z, x) \wedge \neg member(z, y) \rightarrow p(x, y)$.

2.5 Planning

Planning is a branch of Artificial Intelligence (AI), concerned with developing algorithms capable of automatically generating sequence of actions that can realize goals. The main challenge in planning is scalability, since planning in general case is computationally intractable. Planning problems range from those where each action has deterministic effects and complete knowledge about the problem setting is given, to those where actions have stochastic effects and the problem setting is partially known. The problem studied in this work complies to a specific class of planning problems, namely *classical planning*.

Classical planning is characterized by deterministic actions and full knowledge about the environment. As customary in planning, the task is to find a sequence of actions, also known as a plan, that can lead a given initial state to a goal state. As such, finding a plan requires a search through the state-space, where each state captures the state of the environment in a moment of time, and actions are used to transform the states to one another via their effects. The computational challenge in classical planning results from the number of states, and its exponential growth as more and more state features (fluents) are considered.

Predominantly, planning problems have been represented using a language called *STRIPS* [29] or variants of it, which is the preferred representation by the AI planning community. However, the situation calculus has also taken this role on occasion. The state-of the-art techniques and methods developed for planning are associated with the works of people in the planning community, based on the STRIPS representation or its variants. The search control in SC-based representation of planning problems takes a different path. In the following sections, we review the efforts of both parties.

2.5.1 State-of-the-art Planning

Stanford Research Institute Problem Solver, abbreviated as STRIPS, is both the name of the problem solver developed in 1970's, and the underlying language used for representing classical planning problems where action effects are unconditional. A planning problem in STRIPS is represented using a tuple $< F, O, I, G >$ where

- F is the set of *atoms* or *propositions* of interest.
- O is a set of *operators* or *actions*.
- $I \subseteq F$ represents the initial state.
- $G \subseteq F$ represents the goal.

In more details, each operator $o \in O$ is characterized using three sets, also referred to as lists:

- *Add* list denoted $Add(o)$, describes the atoms that the action o makes true.
- *Delete* list denoted $Del(o)$, describes the atoms that the action o makes false.
- *Preconditions* denoted $Pre(o)$, describes the preconditions of the action, that is, the atoms that must be true for the action to be applicable.

In STRIPS, a state is represented as a collection of atoms from F, and as such there are $2^{|F|}$ states, where $|F|$ is the number of atoms in the problem. A variant of propositional STRIPS is the SAS^+ formalism [8]. A planning problem in SAS^+ is also represented using a tuple $< V, O, I, G >$, where O, I and G conceptually correspond to their counterparts in the STRIPS formalism. The main difference between SAS^+ and propositional STRIPS is in their representation of a state, that is in V and F. While STRIPS only allows binary propositions in F, SAS^+ formalism includes *state variables* in V, where the values for the variables are from a finite domain. Consider an example from the logistic domain, a benchmark domain in planning. Assume that there is a truck and a cargo, and 9 locations. The cargo can be in the truck, or each of the 9 locations. The truck can be in any of the 9 locations. The F in the STRIPS formalism will include propositions such as *c-in-truck*, to denote that the cargo is in the truck, $c - in - l_1, \ldots, c - in - l_9$, To denote the cargo is in the location 1 to location 9, respectively. Moreover, the propositions $t - in - l_1, \ldots, t - in - l_9$, denote the truck being in locations 1 to 9, respectively. In the SAS^+ formalism however, V includes the variables v_c and v_t, denoting the state of the cargo and the truck, respectively. The v_c takes the values from the domain $\{t, l_1, \ldots, l_9\}$, while v_t's domain of values is $\{l_1, \ldots, l_9\}$. According to the STRIPS formalism, there are 2^{19} states, corresponding to all valuations of 19 propositions. However, it is clear that, if the truck is in a location, it is not in any other location. That is, only one of *t-in-l** propositions is true in a feasible world state. Similarly, only one of the *c-in-** is can be true in a feasible world state. As such, there are only $10 * 9 = 90$ feasible states. The total number of states in the SAS^+ representation coincides with the feasible states. This is because the SAS^+ formalism preserves the structure between the propositions (all the propositions describing the location of the truck are represented with the same state variable), but the STRIPS formalism is unstructured. This note (preserving the structure in a domain) lays the foundation for *causal graph* heuristics, which will be discussed shortly. It is noteworthy that automatic translation from STRIPS to SAS^+ is possible [45].

Planning Domain Definition Language (PDDL) is the standard language with Lisp-like syntax developed for expressing planning problems [73]. It is used as the input language in the International Planning Competition (IPC): all instances of the planning problems must be specified in PDDL. PDDL has at its core the STRIPS formalism, where all actions have simple unconditional effects and both the initial and goal states are conjunctions of the literals. Over time, PDDL has been extended much beyond this simple setting. PDDL provides a syntactically different way of expressing BATs, but the ideas are essentially the same as in SC. For example, consider the PDDL representation of the action *pickup* that was presented as an example in the previous discussion about SC:

```
(:action pickup
 :parameters (?x1 - man ?x2 - box)
 :precondition (and ( not (exists (?x - box)(holding ?x1  ?x)))
       (onTheGround ?x2))
 :effect (and (holding ?x1 ?x2)
      (not (onTheGround ?x2)) ))
```

Notice that in PDDL, the variables are distinguished by a ? character at front, for example $?x1$ represents a variable. The dash " $-$ " is used to assign types to the variables. In the example above, $?x1$ and $?x2$ are defined to be of types man and box respectively. The preconditions and effects of actions in PDDL are logical formulas constructed from the predicates of the domain and logical connectives. Additionally, notice that preconditions and effects of a given action are bundled together, while in the situation calculus they are spread over multiple axioms.

In PDDL, there are separate *domain description* and the *problem description*. The *domain description* mostly introduces the predicates of the domain as well as the actions with their preconditions and effects. The predicates introduced in the domain description might include PDDL derived predicates [90], corresponding to the state constraints or abbreviations in SC. Below you can see the semantically equivalent representation of the abbreviation presented earlier in SC as a PDDL derived predicate:

```
(:derived (hasWorkToDo ?x)
  (and (man ?x) (not (exists (?x1 - box) (holding ?x ?x1)))
    (exists (?y - box)(and (colour ?y blue) (onTheGround ?y))))
)
```

As can be seen, the PDDL derived predicates simply provide a different syntax for representing SC's state constraints. Overall, the domain description in PDDL roughly corresponds to PAs and SSAs in BATs, as well as state constraints, if any. However, the most notable difference between PDDL and BATs is that in SC, the effects of actions on a fluent are specified in a single axiom, namely the fluent's SSA. However, in PDDL, the effects of actions on a single fluent are characterized separately for each action term.

The *problem description* part of PDDL introduces objects, the initial state and the goal, and as such, roughly corresponds to the initial theory D_{S_0} in BATs. The combination of a problem description and the domain description forms an instance of the planning problem.

State-of-the-art methods in classical planning employ heuristic functions that are extracted from the problem structure to expedite finding the plans. The idea is to guide the search to the direction more likely to reach the goal, by computing an estimate of how far a given state is from the goal state. The domain independent heuristics developed for this purpose have proven effective for solving the benchmark domains used by International Planning Competition (IPC). Here, we review some of the notable heuristics used for planning.

Delete List Relaxation

A key development in research regarding planning was realization that search can be expedited by using heuristics that can be obtained from the problem. The idea in general is to impose a light overhead computation to estimate how far a state is from satisfying the goal and what actions are likely to help it in the process. This information can come very handy to narrow down the search when solving planning problems. Since this realization, a lot of effort has been focused on developing effective heuristics that are easily computed (i.e. with light overhead). To make the heuristics computationally cheap, often a *relaxed* version of the problem is used to compute the heuristics. A prominent relaxation that have proven very successful in the competitions is the *delete list relaxation*, where the negative effects of actions are ignored.

Delete list relaxation heuristic was originated with the following rationale. Finding a solution in STRIPS planning in general case is PSPACE-Complete [14], and as such too difficult. Therefore, a simplified version of the problem is used instead, and the information from the solution to the simpler problem is used in solving the original problem. Solving planning tasks where the actions only have positive effects reduces the problem complexity from PSPACE-Complete to NP-Complete [14]. However, optimal solutions to relaxed problems are still difficult to find, since NP-Complete problems are generally regarded as computationally challenging problems. Therefore, an approximation to the solution for the relaxed problem is considered instead, to reduce the complexity of computing heuristic from NP-Complete to polynomial time.

In more details, given a STRIPS planning problem $P =< F, O, I, G >$, delete list relaxation creates a relaxed problem $P^+ =< F, O^+, I, G >$ which differs from P in that the negative effects of the operators O are ignored. In the relaxed problem, due to the empty delete list, atoms are added, but never deleted. The delete list relaxation heuristic is defined as the approximation of the optimal cost of the plan to problem P^+. There are two famous strategies for computing the approximation of the optimal costs in the relaxed problem. One strategy leads to *additive* heuristic, and the other results in *max* heuristic. Both strategies work the same way for the most part. Both strategies build a graph that estimates the cost of achieving a goal literal p from state s. The graph is initialized with literals in state s having cost of zero. Then, for every action a such that p is a positive effect, p is added to the graph and its cost is set by combining the cost of achieving the preconditions of a. The strategies differ in how they combine the costs of preconditions. The additive heuristic adds them, and the max heuristic takes the maximum. Below, each of these strategies is formally elaborated.

In order to simplify the definitions, and for reasons of uniformity, we introduce a new dummy action End with associated cost of zero, whose preconditions $G_1, \ldots, G_n$ are the goals of the problem (the literals in G), and whose effect is a dummy atomic goal G. The heuristics $h(s)$ simply estimates the cost of achieving this dummy goal G from the state s. The additive heuristic h_{add} is defined as

$$h_{add}(s) \overset{def}{=} h_{add}(Pre(End); s), \tag{2.3}$$

where $h_{add}(Pre(a); s)$ is an estimate of the cost of achieving the preconditions of action a from s, defined from the expressions:

$$h_{add}(p; s) \overset{def}{=} \begin{cases} 0 & \text{if } p \in s \\ min_{a \in O(p)}[cost(a) + h_{add}(Pre(a); s)] & \text{otherwise} \end{cases} \tag{2.4}$$

and

$$h_{add}(Pre(a); s) \overset{def}{=} \sum_{q \in Pre(a)} h_{add}(q; s). \tag{2.5}$$

In these expressions, $h_{add}(p;s)$ stands for the estimated cost of achieving the atom p from s, $O(p)$ stands for the actions in the problem that add p, and $h_{add}(Pre(a),s)$ stands for the estimated cost of achieving the preconditions of the actions a from s. Simply put, the additive heuristic considers the cost of action preconditions (and goals) as the sum of the cost of achieving each precondition (goal) in isolation.

The max heuristic h_{max} is defined using the following three equations:

$$h_{max}(s) \overset{def}{=} h_{max}(Pre(End);s). \tag{2.6}$$

$$h_{max}(p;s) \overset{def}{=} \begin{cases} 0 & \text{if } p \in s \\ min_{a \in O(p)}[cost(a) + h_{max}(Pre(a);s)] & \text{otherwise} \end{cases} \tag{2.7}$$

$$h_{max}(Pre(a);s) \overset{def}{=} max_{q \in Pre(a)} h_{max}(q;s). \tag{2.8}$$

The additive heuristic h_{add} and the max heuristic h_{max} are defined very similarly. The equations 2.6 and 2.7 are produced from the equations 2.3 and 2.4, by replacing h_{add} with h_{max}, respectively. The main difference is in the equations 2.8, where $\sum$ in 2.5 has been replaced with max. The max heuristic, unlike the additive heuristic, considers the cost of action preconditions (and goals) as the maximum of the cost of achieving each precondition (goal) in isolation. The $cost(a)$ in equations 2.4 and 2.7 represents the cost associated to action a, and in general case can be any numeric value. In the case that all actions have uniform costs, the $cost(a)$ for all actions is assumed to be 1. The latter is what we assume in the rest of this paper.

To demonstrate how the additive and max heuristics work, consider the relaxed planning problem $P^+ =< F, O^+, I, G >$, where

- $F = \{P1, P2, P3, P4, P5, P5, P7\}$.
- $O^+ = \{A1, A2, A3\}$, such that
  - $Pre(A1) = \{P1, P2\}$ and $Add(A1) = \{P5\}$.
  - $Pre(A2) = \{P3, P4\}$ and $Add(A2) = \{P6\}$.
  - $Pre(A3) = \{P5, P6\}$ and $Add(A3) = \{P7\}$.
- $I = \{P1, P2, P3, P4\}$.
- $G = \{P7\}$.

In this example, we want to compute the additive and max heuristic values for the initial state S_0, that is $h_{add}(S_0)$ and $h_{max}(S_0)$. The computations are as follows:

$$\begin{aligned}
h_{add}(S_0) &= h_{add}(Pre(End); S_0) \quad \text{from 2.3} \\
&= h_{add}(P7; S_0) \quad \text{from 2.5} \\
&= 1 + h_{add}(Pre(A3); S_0) \quad \text{from 2.4} \\
&= 1 + \big(h_{add}(P5, S_0) + h_{add}(P6; S_0)\big) \quad \text{from 2.5} \\
&= 1 + \Big(\big(1 + h_{add}(Pre(A1); S_0)\big) + h_{add}(P6; S_0)\Big) \quad \text{from 2.4} \\
&= 1 + \Big(\big(1 + (h_{add}(P1; S_0) + h_{add}(P2; S_0))\big) + h_{add}(P6; S_0)\Big) \quad \text{from 2.5} \\
&= 1 + \Big(\big(1 + (0 + 0)\big) + h_{add}(P6; S_0)\Big) \quad \text{from 2.4} \\
&= 1 + \Big(\big(1 + (0 + 0)\big) + h_{add}(Pre(A2; S_0))\Big) \quad \text{from 2.5} \\
&= 1 + \Big(\big(1 + (0 + 0)\big) + \big(1 + h_{add}(P3; S_0) + h_{add}(P4; S_0)\big)\Big) \quad \text{from 2.4} \\
&= 1 + \Big(\big(1 + (0 + 0)\big) + \big(1 + 0 + 0\big)\Big) \quad \text{from 2.4} \\
&= 3.
\end{aligned}$$

$$\begin{aligned}
h_{max}(S_0) &= h_{max}(Pre(End); S_0) \quad \text{from 2.6} \\
&= h_{max}(P7; S_0) \quad \text{from 2.8} \\
&= 1 + h_{max}(Pre(A3); S_0) \quad \text{from 2.7} \\
&= 1 + max\big(h_{max}(P5, S_0), h_{max}(P6; S_0)\big) \quad \text{from 2.8} \\
&= 1 + max\Big(\big(1 + h_{max}(Pre(A1); S_0)\big), h_{max}(P6; S_0)\Big) \quad \text{from 2.7} \\
&= 1 + max\Big(\big(1 + max(h_{max}(P1; S_0), h_{max}(P2; S_0))\big), h_{max}(P6; S_0)\Big) \quad \text{from 2.8} \\
&= 1 + max\Big(\big(1 + max(0, 0)\big), h_{max}(P6; S_0)\Big) \quad \text{from 2.7} \\
&= 1 + max\Big(\big(1 + max(0, 0)\big), h_{max}(Pre(A2; S_0))\Big) \quad \text{from 2.8} \\
&= 1 + max\Big(\big(1 + max(0, 0)\big), \big(1 + max(h_{max}(P3; S_0), h_{max}(P4; S_0))\big)\Big) \quad \text{from 2.7} \\
&= 1 + max\Big(\big(1 + max(0, 0)\big), \big(1 + max(0, 0)\big)\Big) \quad \text{from 2.7} \\
&= 2.
\end{aligned}$$

These calculations, as well as the problem setting is visualized compactly in Figure 2.1, where the estimated cost of each literal is displayed in front of it.

It is worth noticing that in these heuristics, the sub-goals are assumed to be independent, meaning each of them can be achieved with no side effects. This is a wrong assumption in a general setting, but the purpose of the heuristics is not to model the world accurately, rather to provide a sense of direction during search. Also, notice that h_{add} can overestimate the cost of achieving the goal state, as well as can underestimate it. This is because the cost of achieving two atoms can be higher or lower than sum of achieving them separately. As such, h_{add} is not *admissible*, meaning it can overestimate a cost. h_{max} on the other hand, is admissible, since the cost of achieving several atoms from a state can never be lower than the cost of achieving one of them. However, the admissibility of h_{max} comes at the expense of ignoring all but one of the atoms of each action precondition. As such, potentially a lot of useful information is lost in h_{max}. h_{add} improves h_{max} in this sense by including every action precondition in the computation. Next, we provide a brief review of some of the works related to delete list relaxation heuristic.

One of the early efforts for directing search when solving planning problems appears in [72], which uses *regression-match graphs* for computing heuristics. The idea in this work is based on means end analysis, which

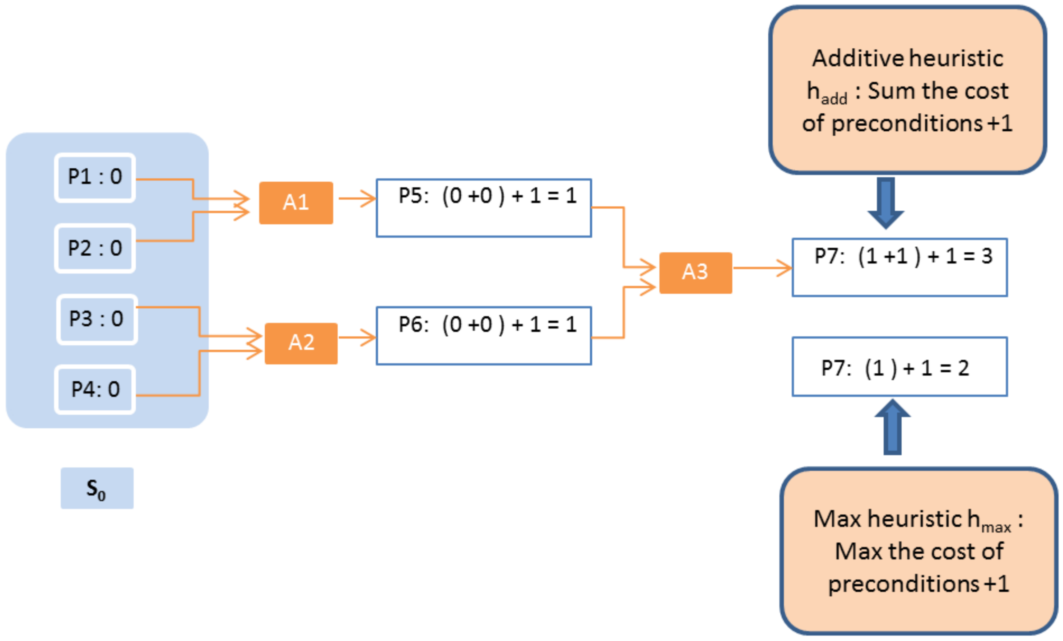

Additive heuristic h_{add} : Sum the cost of preconditions +1
P1 : 0
P2 : 0
P3 : 0
P4: 0
S_0
A1
A2
P5: (0 +0) + 1 = 1
P6: (0 +0) + 1 = 1
A3
P7: (1 +1) + 1 = 3
P7: (1) + 1 = 2
Max heuristic h_{max} : Max the cost of preconditions +1

Figure 2.1: h_{add} and h_{max} for the example

refers to appending an action to the list of partial plan that achieves some unsatisfied goal in the list of goals for the problem. Intuitively, a regression-match graph is a graph that regresses the goal state to the leaves, which are either satisfied in the initial state, or are unsatisfiable. In the context of means end analysis, the regression-match graph is used for estimating the effort for satisfying each goal conjunction. The solution, as admitted by the author, is not complete and not all classical planning problems can be solved. The estimated effort is not so accurate partially since it ignores destructive effects of plan steps on different goal literals.

In [12], Bonet and Geffner study the additive heuristic in variations of the HSP planner. The authors combine the heuristic with forward and backward search, with different search algorithms, and compare and analyze the results. They report that a main bottleneck in forward search is the computation of the heuristic from scratch in every new state. By performing the search backward, the heuristic values can be re-used, and thus backward search is more computationally efficient. However, unreachable states can be introduced that can require a lot of useless search. What is worse is that, it is computationally expensive to identify all of these spurious states, and usually a subset of them is identified. The authors also claim that two advantages of forward planners over regression planners are that the former planners do not generate spurious states and they often benefit from the additional information obtained by the re-computation of the atom costs in every state. The additive heuristic has two main disadvantages: it ignores the positive interactions between subgoals, ignoring the possible simplification of a subgoal by achieving another. Also, since it considers the relaxed problem, it ignores the negative interactions between subgoals.

Another planner that attracted a lot of attention at the time of its emergence was Fast Forward (FF) planner [49, 51]. Similar to HSP, FF uses forward search guided by heuristics obtained from the relaxed problem. However, its heuristic evaluation method takes into account the positive interaction between (sub-)goals, something that HSP's method ignores. As a result, FF's heuristic is more accurate and does not overestimate the cost of goals that their precondition is satisfied by other actions. It also slightly improves the hill-climbing search algorithm used in HSP, as well as identifies a set of useful actions, that are likely to help solve the planning problem.

In more details, FF's heuristic evaluation differs from HSP, in that, it does not simply add the cost of the facts in a goal in order to obtain the goal's cost. Instead, it uses a more sophisticated procedure, inspired by Graphplan [10], where first a planning graph of alternating fact and action layers is constructed, starting from the current state and ending in a fact layer where all the goals are contained. Each action layer consists of all the possible actions in the preceding fact layer, and adds the positive effects of the actions to the proceeding fact layer. In the second stage, the planning graph is processed to calculate relaxed plans, as follows. Starting from the top most fact layer containing all the goals, if a goal is contained in the previous fact layer, then delay its consideration until that layer is processed. Otherwise, select the action that satisfies this goal, and add the preconditions of that action as a goal that needs to be achieved in the previous fact layer. Once all the goals at the current fact layer are processed, continue the same procedure for the goals in the preceding goal layer, until we reach the first layer of the graph. This will result in a plan $< O_0, O_1, ..., O_{m-1} >$ (m here is the depth of the planning graph) where each O_i contains the actions that were chosen when considering the ith layer. The FF's heuristic for each plan is calculated by counting the actions in the relaxed plan. It is important to note, that by delaying processing goals that can be achieved in the preceding layers, the relaxed plan processes the goals only once, which in turn takes into account the positive interaction between goals. This is why FF's heuristic does not overestimate the cost of goals when achieving a sub-goal simplifies the achievement of the other. FF also benefits from search pruning techniques which are derived from examining the relaxed plans, such as identifying helpful actions.

Abstraction and Causal Graphs

Apart from the delete list relaxation, the ideas based on abstraction have also been used to simplify the planning problems. The intuition behind abstraction is to simplify the problem's state space by ignoring certain details.

Formal definition of abstraction depends on formal definition of state-space. A state-space is a directed graph with labelled and weighted edges, in which vertices represent states and edges represent state transitions. The label of each edge represents the name of the action that causes the transition, and the weight corresponds to the cost of the action. A search problem is the following: given a state-space S, an initial state I_s, and a set of goal states G, find a minimum-cost path through S from I_s to some $g \in G$. Formally, an abstraction is a homomorphic mapping φ from the states of a state-space S to some abstract state-space AS, such that all labelled paths and goal states are preserved. That is, for all edges $s \xrightarrow{a:c} s'$ in S, $\varphi(s) \xrightarrow{a:c'} \varphi(s')$ is in AS such that $c' \leq c$, and if $s \in G_S$, then $\varphi(s) \in G_{AS}$. Homomorphism requirement ensures φ is the minimum mapping satisfying the above requirements.

In general, two approaches have employed abstraction for solving planning problems. One approach, uses abstraction as *refinement*, and the other as *heuristic*. In the refinement approach, solutions to the abstract problem are extended into concrete solutions. In the heuristic approach, the solutions in the abstract problem are used to compute heuristic values for the original problem. Here, we briefly elaborate on the core of each of these approaches.

In refinement approach, a plan in the abstract state-space is used as a skeleton for the original plan. That is, the plan for the abstract problem serves as a high-level solution to the concrete problem, and all that remains is to account for the details of the steps in the solution. In this approach, the abstract solution decomposes the original problem into a sequence of sub-problems demanding to be solved in the original state-space. Refining an abstract plan is usually accomplished by inserting additional operators between the operators in the abstract solution. If there is a solution to the sub-problems in the original state-space, the solution to the overall problem has been achieved. Otherwise, more planning in the abstraction space is required to discover an alternative solution. In this approach, the concrete problem is only consulted when there is evidence that the abstract problem can be solved. This approach easily generalizes to multiple abstraction levels, i.e. a hierarchy of abstractions, where abstract levels are refined in order from the most abstract to the least, terminating at the original concrete level.

The works in [78] and [75] utilize the same idea in applications such as finding proofs in symbolic logic and general problem solver. [83] applies this method to planning problems and reports promising experimental results, in which the planner developed based on this idea outperforms the planners that work only on the original problem. The advantage of this approach is more evident in the case when the number of operator in a plan increases. In one of the first attempts to formalize good abstractions, meaning those that are likely to be refinable in the concrete state-space, Tenenberg introduces the concept of *upward solution property* [89]. Informally, upward solution property demands that if there exists a concrete solution, there also should exist an abstract solution. Although this property is intuitively clear, it is vital for success of this approach. To improve on properties of desire for abstractions, the *ordered monotonicity property* is introduced in [54]. The ordered monotonicity property signifies good abstractions by imposing a relationship between abstract and concrete plans: any concrete plan should be derived from an abstract plan such that all actions in the abstract plan are left intact and relevant to the concrete plan after refinement. The ordered monotonic property guarantees that every concrete solution is a natural extension of an abstract solution, but falls short of guaranteeing that every abstract plan can be refined into a concrete plan. The *downward refinement property* introduced by Bacchus and Yang [6], is a property that guarantees the latter. The downward refinement property guarantees that an abstract plan can be refined to a concrete-level solution without backtracking across abstraction levels. However, in the actual planning domains, this property is hardly realized. As such, approximating it is more practical. In fact, the authors in [7] provide an analytic model where probability of refinability of an abstract plan is considered, such that if the probability is 1, then the downward refinement property is guaranteed.

In the abstraction heuristic approach however, a heuristic search in the original state-space is performed, where heuristics are computed in the abstract state-space. Abstraction heuristics tend to take the heuristic value as the optimal cost of the solution to an abstraction of the planning task. As such, the abstract state-space must yield computationally easier problems. The main challenge in this approach is how to define a good abstraction automatically, such that it simplifies the problem considerably and yet the solutions in the abstract space are informative enough for the original problem. Different strategies for defining the abstraction exist, each resulting in a different heuristics. Examples of the strategies to define the abstraction are to ignore some of the state variables of the problem (an example is pattern databases heuristic), or to “collapse” several states into one (an example is merge-and-shrink heuristic).

Pattern databases [22], abbreviated PDB, is the heuristic developed using abstraction when some of the state variables of the problem are completely ignored. The state variables that are not ignored define a *pattern*. PDB exhaustively computes costs for all abstract states and creates a look-up table mapping abstract problem states to cost estimates for the original problem states. The heuristics obtained in this way have proven successful for many domain, but suffer some drawbacks. In more details, other than the potentially long time required for preprocessing, pattern databases are suitable when there is one single goal, as the procedure for creating them is very inefficient when multiple goals exist. Additionally, it consumes large amounts of memory for storing the database, which is a waste if planning is not done regularly, as a tiny portion of it is usually used during the search.

Hierarchical heuristic search algorithms, unlike PDB, search the abstract state-space only when needed, and as such they address some of the shortcomings of PDB. However, their search algorithms are inferior to methods in PDB, as an abstract nodes might get expanded multiple times during a search for a path. [58] addresses this limitation of hierarchical heuristic search, by avoiding the redundancies in node expanding during searching in an abstract level. The authors in [48] describe abstraction strategies that improve on performance of pattern databases heuristic, as well as include it as a sub-case. The work in [46] studies the connection between the common heuristics used in planning, including delete list relaxation and abstraction heuristics. [11] proposes a new abstraction heuristic, that builds on merge-and-shrink heuristics by capturing quantitative aspects of the problem (such as the number of unsatisfied goals in a state) and yields more informative heuristics. Finally, [9] studies the connection between abstraction as refinement and abstraction as heuristic.

The work on developing efficient heuristics for planning problems is not limited to what is mentioned above. In [43], Helmert emphasizes the importance of the causal structure of the domain for finding effective heuristics. Therefore, he relies on *causal graphs* for finding heuristics and presents an algorithm for finding plans and detecting dead-ends. Causal graphs signify how components of a domain depend on each other, and thus have domain analysis value as well (in general, the more dependency exists, the more complex the domain is). A successful planner that utilizes the causal graph heuristics is Fast Downward [44]. [37] proposes a novel algorithm that combines the causal graph heuristic proposed by Helmert with additive and max heuristics. [47] redefines the previously procedurally defined causal graph heuristics declaratively using mathematical equations and claims that the causal graph heuristic is the additive heuristic plus contextual information. The authors of [53], in order to avoid pitfalls common in delete relaxation, partition the variables in the planning problem into two sets: one which delete relaxation will be effective for (red variables), and the other where regular semantics are applied to (black variables). This work elaborates on how to select the variables appropriately and how to generate plans in the red-black planning problems. The work of Hoffmann in [50] is dedicated to studying the relation between delete list heuristics used in FF and causal graphs developed for planning problems. This work results in an automatic domain analysis method that can distinguish between easy and hard domains as well as effectiveness and usefulness of the heuristic function by studying the properties of the causal graph. There are many more works regarding heuristics, but their review fits best in a different paper.

2.5.2 Planning in SC

The SC has been mostly associated with the planning problems in AI. The SC gives a logical account of planning by providing a specification of the planning task. Given a BAT that includes an axiomatized initial situation, and a goal statement, find a sequence of actions that will lead to a situation in which the goal will be true. Below we briefly review the relations between STRIPS and SC, as well as efforts for search control in SC.

As noted in [61], STRIPS language at the time of its emergence did not have a well-defined semantics and it appeared that seemingly harmless modifications in the representation of the problem caused incorrect plans to be produced. As a result, extensive work on defining precise semantics for STRIPS and its characteristics, and planning in general was initiated.

John McCarthy attempted to formalize STRIPS in SC in 1985 [71] by introducing predicates in SC for characterizing STRIPS operators, namely precondition, delete and add predicates. He also introduced a single axiom in SC for STRIPS illuminating the effect of the actions on database of propositions known to be true in a given situation (the SC actions correspond to STRIPS operators and the effects correspond to add and delete list of the operators). Sierra-Santibanez in [85] built on John McCarthy's formulation of STRIPS in SC, and expanded it with formalizing reasoning strategy used in STRIPS in SC, as well as heuristic and search strategy.

Soon after McCarthy's unfinished attempt, Lifschitz proposed to define semantics for operator descriptions for sound STRIPS systems [61]. In order to solve the problem of defining under what conditions the delete list of an operator is sufficiently complete (ensuring all the formulas that should be deleted after execution of an action are deleted), he restricts the add and delete lists of the operators to atomic sentences, i.e. single predicates, and restricts the world models to include only atomic sentences. Non-atomic formulas are allowed in world models if they are always true (are never deleted). Moreover, a non-atomic formula is allowed in add lists if the formula is satisfied in every state of the world. The main drawback of his approach is in being meta-theoretic, and consequently it remains unclear what should be done when a logical theory is incomplete (when having incomplete knowledge). Lifschitz's work provides the foundation of the semantics of PDDL [73]. Later, other researchers provided semantics for more complicated features of PDDL, such as numeric and temporal extensions [30].

In 1997, Lin and Reiter [63] provided declarative semantics for STRIPS operators in terms of progression of initial theory in SC. Moreover, they proved that planning with open world assumption and incomplete knowledge that conforms to certain structural restrictions (strongly context-free successor state axioms) can be done correctly in SC. The main advantage of their approach, other than being general, is that it departs from meta-theoretic approaches used previously for defining semantics for STRIPS.

The idea of working with incomplete knowledge is further explored in [65], where Liu and Levesque examine the complexity of query answering in "proper knowledge bases", which allow for a limited form of incomplete knowledge. What is particularly interesting in this work is that reasoning with incomplete knowledge bases is demonstrated in terms of database techniques, which is not less efficient than query answering from traditional knowledge bases. Further, Liu makes it apparent in her thesis that if we have complete knowledge about the contexts of any context-dependent action, then we have sufficient knowledge for completeness of progression of proper KBs. This means we know whether a ground atom holds or not after executing some action. This results in progression of a proper KB to remain in a proper KB, which in turn makes possible iterative progression as a form of reasoning on proper knowledge bases. In her thesis, Liu proposed a tractable, sound, and sometimes complete solution to the projection problem in the presence of context-dependent actions and incomplete first order knowledge in the form of proper KBs. The projection problem is prerequisite for planning; it consists in answering whether a given logical query formula holds after executing a sequence of ground actions.

Elaborating on the notion of progression [64], Liu and Lakemeyer introduce an efficient procedure for computing progression, and prove that for local-effect actions progression is always first order definable. Moreover, they present an efficient procedure for computing progression for local-effect action in $proper^{+}$ knowledge bases (which allow disjunctions between predicates in the rules whose left hand sides are boolean combination of equality formulas).

In effort to integrate the achievements of the planning community with those of action programming languages, authors of [19] integrated Golog, as an action programming language based on SC, with the developments from the planning community. More specifically, in order to improve the performance of Golog programs that include planning sub-tasks, they translate the planning sub-tasks in a Golog program into PDDL and use existing planners for solving the sub-tasks. By doing so, they use SC to provide declarative semantics for different fragments of PDDL. In other words, concerning solving planning problems, they use SC for providing semantics of PDDL, and the actual plans are generated using the planners that take as input a PDDL planning task.

The importance of derived predicates, also known as state constraints or abbreviations, is argued in [90]. The semantics provided for derived predicates in PDDL is based Lifschitz's work and as so is provided by grounding them. Other semantics for restricted form of state constraints is presented in [74], where state constraints are compiled into the SSAs. Other research in this regard includes the work of Lin and Reiter [62], where general case of state constraints are considered, and therefore they are compiled as second-order formula in the SSAs.

2.6 Computer Assisted Organic Synthesis

In the recent years, there has been an ongoing progress in developing new methodologies or expanding existing methodologies in order to employ computers to the task of solving organic synthesis problems. Using these methodologies, chemists can plan the chemical reactions needed to take place for a target to be synthesized. Several methods have been developed as a result of the efforts in this area. The efforts in this regard fall under the umbrella of Computer Assisted Organic Synthesis (CAOS). In this section, we give a brief overview of some of the methods and systems developed for CAOS, while elaborating one of them in more details since it is used for comparison with the methodology developed in this work. This review is by no means comprehensive; we concentrate only on some of the well known methods and systems. There are a number of papers that provided detailed and comprehensive reviews [91, 16, 20, 77, 88, 33, 94, 76, 39].

2.6.1 Brief Review of Some CAOS Systems

The first CAOS system was organic chemical simulation of synthesis (OCSS) developed by Corey and Wipke [21, 26]. OCSS uses the retrosynthetic analysis approach, that is, it starts from the target molecule and goes backward regressing it to starting material. Retrosynthesis attempts to simplify the target molecule by transforming it recursively to simpler molecules until a set of available starting materials has been reached. Key concepts in retrosynthesis are *transform* (reverse of a reaction) and *retron* (the minimal precondition making a transform applicable). Corey's system uses functional groups and topological structures in the molecules as the basis of its procedure. OCSS benefits from heuristics developed from analysis of field of organic chemistry, generally in the form of generic principles governing organic synthesis. SYNCHEM was introduced by Gelernter in 1973 [35, 34, 16, 20]. SYNCHEM is a non-interactive system designed to utilize machine learning (ML) techniques from Artificial Intelligence to solve complex synthesis design problems. SYNCHEM's goal was to develop a system of programs to give correct synthesis routs of organic structures non-interactively. SYNCHEM was equipped with a sufficiently developed knowledge base and with heuristic search control which resulted in relatively good performances. SYNCHEM also left the opportunity open for refining its search-guidance strategies and tactics for further improvement. SYNCHEM works by analyzing the target molecule to identify functional groups and structural patterns, which are then used along with its reaction library and chemistry heuristics to recursively produce synthesis routes.

Retrosynthesis-based Assessment of Synthetic Accessibility (RASA) is a system used for computer aided drug discovery (CADD) [52]. RASA is based on Corey's retrosynthetic analysis, but employs a series of strategies to limit its probable combinatorial explosion in the synthesis tree. Some of these strategies include declaring a limit for the synthesis tree's depth, omitting branches that include unstable molecules and those which result in substantially more complex precursors. Route Designer [59] is a non-interactive system that works based on retrosynthetic analysis and automatically generates rules describing possible retrosynthetic transformations from reaction databases. To extract retrosynthetic transformation rules, the functional groups influencing the reactions in the database are identified, and similar reactions are clustered in the same group. Subsequently, template rules describing the generic cluster are generated which are refined to produce completed reaction rules. In order to deal with large transformation rules and probable combinatorial explosion, it employs backward search with a number of heuristics and user defined limits. These include search depth bound and unbreakable bonds (or must-break bonds) in the starting materials. Lastly, SYMBEQ is a computer program based on Formal-Logical Approach [92] which performs systematic search on symbolic equations describing reactivity of organic compounds [98]. The Formal-Logical Approach aims at reaction design as opposed to organic synthesis. It uses formal algebraic models to search for potentially new types of chemical reactions. In the output of SYMBEQ program, replacing abstract symbols by chemical atoms in accordance to the allowed valence can result in new chemical reactions, which their correctness can be verified theoretically and practically.

2.6.2 Synthesis via Proof Number Search

In a recent effort, Heifets and Jurisica [41] utilize proof number search for finding a correct synthesis route that can synthesize a goal molecule from a library of commercially available chemical compounds. In this work, the organic synthesis problem is formulated as a two-player zero-sum game, and the synthesis route is discovered by proof-number search on an AND/OR graph. AND/OR graphs are composed of two types of nodes, namely AND-Nodes and OR-Nodes. An AND-Node is marked proven if all its children are proven, and an OR-Node is marked proven if any of its children are marked proven. Moreover, the AND/OR graphs are directed graphs with alternating layers of AND and OR nodes, where AND-Nodes are connected to OR-Nodes, and vice versa. In this work, the OR-Nodes in the AND/OR graph represent molecules, the edges from OR-Nodes to AND-Nodes represent the chemical reactions that can produce the molecules encoded in the OR-Nodes, and the AND-Nodes denote the reagents of the reactions leading to their parent OR-Node molecule. Each AND-Node is expanded to OR-Nodes, each representing one of the reagents that the AND-Node is encoding. The proof number search simply searches for a way to mark the root node proven by expanding the nodes with the minimum proof number first.

In more details, in [41] the goal molecule is put in an OR-Node in an AND/OR graph, and all the reactions from the reaction library are checked against it to derive the reactions $R_1, \ldots, R_n$ capable of resulting in the goal molecule. At this stage, the effects of the chemical reactions are applied in reverse to the goal molecule to derive the reagents of the reactions capable of producing the goal molecule. These reagents are put in AND-Nodes of the corresponding reaction edge. This step conceptually translates to: the goal molecule (OR-Node) can be synthesized if R_1 is executed, or if R_2 is executed,. . ., if R_n is executed. The AND-Nodes are then expanded to multiple OR-Nodes, each denoting a reagent required for the reaction to happen. Conceptually, this step means for the reaction to be applicable, all the reagents (AND-Node) must be present. This process is continued and each time the OR-Nodes are checked against a library of starting material, and marked proven if they exist in the library. This procedure is guided by the standard proof-number heuristics, so that the nodes with the least estimates are expanded first.

The authors also report the performance of their method on a library of benchmark problems and make the benchmark publicly available. The library of benchmark problems is composed of 20 organic synthesis problems, taken from university level midterms for Chemistry students. However, not all reactions in the library have their full effects specified. Moreover, most of the reactions in the library are unbalanced. This set of benchmarks is publicly available from [2]. It is noteworthy that no knowledge representation framework for representing molecules and reactions was developed in [41]. The proof number search algorithm from [41] delegated all subgraph isomorphism queries and other chemistry reasoning to the proprietary software developed by ChemAxon [15]. For this reason, all experimental results reported in [41] significantly depend on the quality of ChemAxon's software.

Chapter 3

Representing Organic Chemistry in SC

This chapter discusses how chemical molecules and reactions are modeled in the Situation Calculus BATs augmented with abbreviations, and how they are reasoned about. More specifically, two approaches are described: namely the *micro* approach and the *macro* approach. The two approaches share how they represent molecules, and differ in how they model chemical reactions. The micro approach models a chemical reaction by explicitly representing each step of the process in terms of changes of the bonds between the atoms involved. In contrast, the macro approach compactly represents a chemical reaction as a whole without having to worry about the process too much.

The rest of this chapter is organized as follows. First, representing molecules in SC is explained. Then, the subsequent section uses the representation of the molecules to describe modeling of chemical reactions in SC, with separate discussions about the micro and macro approaches.

3.1 Representing Molecules

3.1.1 The Basics

Molecules are often modeled as undirected graphs, where the vertices of the graph represent the chemical atoms in the molecule, and edges represent the bond between the atoms. The same intuition is used here for modeling molecules in SC. The atoms are modeled as SC objects, and the bonds as relations over the objects, or in SC terms, as fluents.

More specifically, atom types are decided using situation-independent predicates, corresponding to the name of the atoms in the periodic table. For example, $Carbon(C)$ defines the object C to represent a carbon atom, or $Hydrogen(H)$ asserts object H to be a hydrogen atom.

The bonds between atoms are modeled using fluents, so that their truth value can change as the situations change. This is critical as it allows reasoning about chemical reactions. For each type of bond between chemical atoms, a fluent is introduced. Examples are $Bond(X, Y, S)$, $DoubleBond(X, Y, S)$, $TripleBond(X, Y, S)$ and $AromaticBond(X, Y, S)$. The fluent $Bond(X, Y, S)$ is true, if the atoms X and Y share a single bond in the situation S. The other fluents for bonds are defined similarly, with the name of the fluents corresponding to what type of bond they represent.

The BAT's initial theory D_{S_0}, specifies the molecules in the initial state. It includes formulas that are built out of the described ingredients and logical connectors and quantifiers. For example, a water molecule H_2O in the initial situation S_0 can be represented as

$$\begin{aligned}&Hydrogen(H') \wedge Hydrogen(H'') \wedge \quad H' \neq H'' \wedge Oxygen(O) \wedge\\ &\quad \forall x(Bond(x, O, S_0) \rightarrow (x = H' \vee x = H'')) \wedge\\ &\qquad \forall y(Bond(H'', y, S_0) \rightarrow y = O) \wedge \forall y(Bond(H', y, S_0) \rightarrow y = O).\end{aligned}$$

The above formula asserts existence of hydrogen atoms H' and H'', as well as O as an oxygen atom. It also asserts that in the initial situation S_0 atoms H' and H'' have a single bond with O, and no other bond. Additionally, O has only bonds with H' and H''.

3.1.2 Abbreviations

The main purpose of abbreviations are to identify existence of chemical classes, functional groups or specific molecules in a situation. Abbreviations are simply definitions of important chemical concepts, corresponding to derived predicates in PDDL. Here, some examples of abbreviations defining functional groups, chemical classes, and specific molecules are presented, followed by a general discussion about the abbreviations.

Abbreviation Examples

Although most of the abbreviations are situation dependent, there are some situation-independent abbreviations as well, which are used to introduce groups of atoms. An example is $Halogen(x)$, which is true only if x is an atom belonging to the halogen group in the periodic table. The defining abbreviation for $Halogen$ would be as follows:

$$Halogen(x) \overset{def}{=} Fluorine(x) \vee Chlorine(x) \vee \ldots$$

Another example is $Atom(x)$, which is true if x is any chemical atom:

$$Atom(x) \overset{def}{=} Carbon(x) \vee Oxygen(x) \vee Hydrogen(x) \vee \ldots$$

As for examples of situation-dependent abbreviations, the hydroxyl functional group $-OH$, can be defined using the following abbreviation:

$$\begin{aligned}&Hydroxyl(o, h, s) \overset{def}{=} Oxygen(o) \wedge Hydrogen(h) \wedge Bond(o, h, s) \wedge\\ &\quad \exists^{=2}\, x(Atom(x) \wedge Bond(o, x, s)) \wedge \exists^{=1}\, x(Atom(x) \wedge Bond(h, x, s)).\end{aligned}$$

In the above formula, $\exists^{=1}$ ($\exists^{=2}$, respectively) is a counting quantifier saying that there exists exactly one (there are exactly two, respectively) entities for which quantified formula holds. The counting quantifiers can be replaced with usual (but less readable) first order logic syntax. For example, $\exists^{=2}\, x(\varphi(x))$ stands for $\exists x_1 \exists x_2(\varphi(x_1) \wedge \varphi(x_2) \wedge x_1 \neq x_2 \wedge \forall y(\varphi(y) \rightarrow (y = x_1 \vee y = x_2)))$.

Next, consider the abbreviation for a chemical class, namely alcohol R$-$OH in its most generic form, which is a hydroxyl group single bonded to a carbon atom that has three other single bonds. In the below definition for alcohol, the atoms o and h which appear as the arguments of the abbreviation identify the whole alcohol molecule, without having to account for every other atom in the molecule.

$$Alcohol(o, h, s) \overset{def}{=} Hydroxyl(o, h, s) \wedge \exists c(carbon(c) \wedge Bond(o, c, s) \wedge \exists^{=4}\, x(Bond(c, x, s))).$$

It is worth noting that the arguments of the abbreviations, also known as the *key atoms*, are often chosen from the atoms at the common reaction sites, but need not necessarily be so. So it is possible to define an alcohol molecule as $Alcohol(c, o, s)$, where the key atoms are the carbon attached to the hydroxyl, and the oxygen of the hydroxyl. In general, the key atoms of the abbreviation, although are only a part of the molecule, represent the whole molecule in a situation. In the $Alcohol$ abbreviation, if atoms o and h are found that satisfy the requirements of the abbreviation, then they are necessarily part of an alcohol molecule, and as such represent the molecule.

We can also define arbitrary chemical classes, those that are not necessarily named by chemists. Let us assume we need to find molecules that have a fluorine atom and an oxygen atom somewhere in their structure. A prerequisite to achieving this, is to be able to determine whether two atoms belong to the same molecule, i.e. are connected to each other with chemical bonds either directly or indirectly. To achieve this, we introduce the abbreviation $Connected(x, y, s)$, which is the recursive transitive closure relation defined on bond fluents. For the sake of simplicity, if there are single bonds only, it is defined as follows:

$$Connected(x, y, s) \overset{def}{=} Bond(x, y, s) \lor \\ \exists z(Bond(x, z, s) \land Connected(z, y, s)).$$

Then, the abbreviation for molecules containing fluorine and oxygen will be as follows:

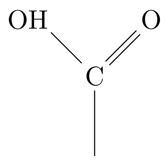
$$MoleculeContainingFluorineAndOxygen(f, o, s) \overset{def}{=} \\ Fluorine(f) \land Oxygen(o) \land Connected(f, o, s).$$

The abbreviation $Connected$ comes handy when defining other chemical concepts that need to consider all the atoms in a molecule. Some examples are hydrocarbon, which are compounds consisting entirely from carbon and hydrogen atoms, and inorganic molecules, i.e. molecules that do not contain any carbon atoms. Consider these concepts defined as abbreviations below:

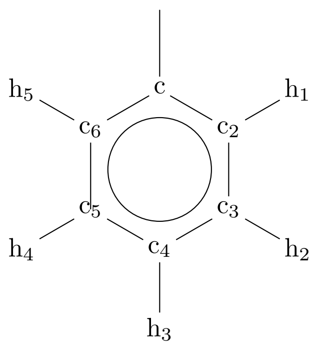
$$Hydrocarbon(c, h, s) \overset{def}{=} Carbon(c) \land Hydrogen(h) \land Connected(c, h, s) \land \\ \neg\exists x(Connected(c, x, s) \land \neg Hydrogen(x) \land \neg Carbon(x)).$$

$$Inorganic(x, s) \overset{def}{=} \neg Carbon(x) \land \neg\exists c(Carbon(c) \land Connected(c, x, s)).$$

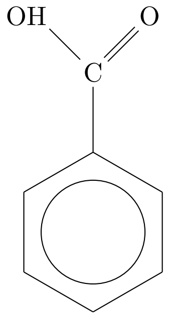
We provided examples of how functional groups and chemical classes can be represented as abbreviations above. Consider an example of an abbreviation for a specific molecule, benzoic acid below. The structure of benzoic acid is demonstrated below along with two of its sub-molecules: carboxyl group and phenyl group. The abbreviation for benzoic acid uses the abbreviations of carboxyl group and phenyl. Phenyl has a cyclic structure formed from carbon atoms with aromatic bonds with each other, distinguished by a straight line and a curved line. In the structure of phenyl below, we have replaced atoms with variables (lower case letters) to facilitate understanding of its abbreviation below. In phenyl's abbreviation, the predicate UNA ensures that its arguments are pairwise different, and thus provides unique name assumption for the carbon and hydrogen atoms in the abbreviation.



Carboxyl



Phenyl



Benzoic acid

$$BenzoicAcid(c_1, c_2, s) \overset{def}{=} Phenyl(c_1, s) \wedge Carboxyl(c_2, s) \wedge Bond(c_1, c_2, s).$$

$$\begin{aligned}
&Carboxyl(c, s) \overset{def}{=} Carbon(c) \wedge \\
&\quad \exists o_1, o_2, h(Hydroxyl(o_1, h, s) \wedge Oxygen(o_2) \wedge \\
&\quad \exists^{=2}\, x(Bond(c, x, s)) \wedge \exists^{=1}\, x(DoubleBond(c, x, s)) \wedge \\
&\quad \exists^{=1}\, x(DoubleBond(o_2, x, s)) \wedge \\
&\qquad Bond(c, o_1, s) \wedge DoubleBond(c, o_2, s) \wedge o_1 \neq o_2).
\end{aligned}$$

$$\begin{aligned}
&Phenyl(c, s) \overset{def}{=} Carbon(c) \wedge \\
&\quad \exists c_2, c_3, c_4, c_5, c_6, h_1, h_2, h_3, h_4, h_5 \\
&\quad (Carbon(c_2) \wedge Carbon(c_3) \wedge Carbon(c_4) \wedge Carbon(c_5) \wedge Carbon(c_6) \wedge \\
&\quad Hydrogen(h_2) \wedge Hydrogen(h_3) \wedge Hydrogen(h_4) \wedge Hydrogen(h_5) \wedge \\
&\quad \exists^{=2}\, x(AromaticBond(c, x, s)) \wedge \exists^{=2}\, x(AromaticBond(c_2, x, s)) \wedge \\
&\quad \exists^{=2}\, x(AromaticBond(c_3, x, s)) \wedge \exists^{=2}\, x(AromaticBond(c_4, x, s)) \wedge \\
&\quad \exists^{=2}\, x(AromaticBond(c_5, x, s)) \wedge \exists^{=2}\, x(AromaticBond(c_6, x, s)) \wedge \\
&\quad \exists^{=1}\, x(Bond(h_1, x, s)) \wedge \exists^{=1}\, x(Bond(h_2, x, s)) \wedge \exists^{=1}\, x(Bond(h_3, x, s)) \wedge \\
&\quad \exists^{=1}\, x(Bond(h_4, x, s)) \wedge \exists^{=1}\, x(Bond(h_5, x, s)) \wedge \\
&\qquad AromaticBond(c, c_2, s) \wedge AromaticBond(c_2, c_3, s) \wedge AromaticBond(c_3, c_4, s) \wedge \\
&\qquad\quad AromaticBond(c_4, c_5, s) \wedge AromaticBond(c_5, c_6, s) \wedge AromaticBond(c_6, c_1, s) \wedge \\
&\qquad\qquad Bond(c_2, h_1, s) \wedge Bond(c_3, h_2, s) \wedge Bond(c_4, h_3, s) \wedge \\
&\qquad\qquad\quad Bond(c_5, h_4, s) \wedge Bond(c_6, h_5, s) \wedge \\
&\qquad\qquad\qquad UNA(c, c_2, c_3, c_4, c_5, c_6) \wedge UNA(h_1, h_2, h_3, h_4, h_5).
\end{aligned}$$

Consider another example, that of cyclobutane, which is a cyclic structure constructed from four carbon atoms with single bonds with each other, each of which has two distinct hydrogen atoms attached to:

$$
\begin{aligned}
&Cyclobutane(c_1,s) \overset{def}{=} \exists c_2,c_3,c_4,h_1,h_2,h_3,h_4,h_5,h_6,h_7,h_8\\
&\quad\Big(\textstyle\bigwedge_{i=1}^{4} Carbon(c_i) \bigwedge_{i=1}^{8} Hydrogen(h_i) \wedge\\
&\qquad UNA(c_1,c_2,c_3,c_4,) \wedge UNA(h_1,h_2,h_3,h_4,h_5,h_6,h_7,h_8)\\
&\qquad\quad \textstyle\bigwedge_{i=1}^{3} Bond(c_i,c_{i+1},s) \wedge Bond(c_4,c_1,s)\\
&\qquad\qquad \textstyle\bigwedge_{i=1}^{4} (Bond(c_i,h_i,s) \wedge Bond(c_i,h_{i+1},s))\\
&\qquad\qquad\quad \textstyle\bigwedge_{i=1}^{4} \Big(\exists^{=4}\, x(Bond(c_i,x,s))\Big)\\
&\qquad\qquad\qquad \textstyle\bigwedge_{i=1}^{8} \Big(\exists^{=1}\, x(Bond(h_i,x,s))\Big)\Big).
\end{aligned}
$$

Moreover, abbreviations are also used for representing goal molecules of the planning instances. For example,

$$Goal(s) \overset{def}{=} (\exists c_1,c_2) BenzoicAcid(c_1,c_2,s).$$

This formula states that our goal is to reach a situation s in which there are atoms identifying a benzoic acid molecule.

Abbreviations Discussion

As noticeable from the above examples, it is possible to have nested abbreviations. This is useful as it provides modularity in our representation, and can come handy when a chemical class or molecules subsumes other functional groups or sub-graphs that have been previously defined as abbreviations.

Most of the chemical classes, functional groups and specific molecules can be expressed as abbreviations that are constructed from, or can be transformed such that are constructed from fluents, abbreviations, conjunctions, existential quantifiers and negation. More specifically, Lloyd-Topor transformations [66] can be applied to the rules that use $\forall$ or $\vee$ (typically obtained after eliminating counting quantifiers) to convert them into logically equivalent rules without $\forall$ or $\vee$. Take for example the abbreviation for hydroxyl above. Lloyd-Topor transformations can be applied so that the rule be transformed to:

$$
\begin{aligned}
&Hydroxyl(o,h,s) \overset{def}{=} Oxygen(o) \wedge Hydrogen(h) \wedge Bond(o,h,s) \wedge\\
&\quad \exists^{=2} x(Atom(x) \wedge Bond(o,x,s)) \wedge \exists x_1(Atom(x_1) \wedge Bond(h,x_1,s) \wedge \neg P(h,x_1,s)).
\end{aligned}
$$

where $P(h,x_1,s)$ is the auxiliary predicate introduced by one of Lloyd-Topor's transformations. The rule defining $P(h,x_1,s)$ is as follows:

$$P(h,x_1,s) \overset{def}{=} \exists y(Atom(y) \wedge Bond(h,y,s) \wedge y \neq x_1).$$

The remaining counting quantifier $\exists^{=2}$ can be similarly eliminated.

It is interesting to note that these abbreviations are indeed expressible in stratified Datalog$^{\neg}$. This is in part possible because no function symbols are present in these abbreviations, and all the occurrences of $\vee$ and $\forall$ can be eliminated. Consider the Datalog$^{\neg}$ rules corresponding to the above definitions of $BenzoicAcid$ and $Connected$, noting that in Datlog, variables appearing on the right hand side of the rules that do not appear in the head are implicitly assumed to be existentially quantified:

$$BenzoicAcid(c_1,c_2,s) \leftarrow Phenyl(c_1,s), Carboxyl(c_2,s), Bond(c_1,c_2,s).$$

$$
\begin{aligned}
&Connected(x,y,s) \leftarrow Bond(x,y,s).\\
&Connected(x,y,s) \leftarrow Bond(x,z,s), Connected(z,y,s).
\end{aligned}
$$

The stratification of these abbreviations comes from the fact that there is no cyclic (not to be confused with recursive) definition in chemistry. Cyclic definition means interdependent definitions of two compounds, i.e. definition of compound A depends on compound B and definition of compound B depends on compound A. This is due to the fact that abbreviations essentially represent sub-graphs of graphs, and the $IsPartOf$ relation is acyclic. In other words, you cannot find a graph that has a strict sub-graph that includes the graph. This makes it possible to always put the IDB rules that are used in the right hand side of other rules in a stratum before the stratums defining the dependent rules.

For example, in the above examples, the rules defining $Connected$ appear in the same stratum, as allowed in stratification since the IDB relation $Connected$ on the right hand side of the rule is not negated. Similarly, the rules defining $Phenyl$ and $Carboxyl$ appear in a stratum no after the stratum containing the rule for $BenzoicAcid$. On the other hand, in the above examples for inorganic and hydrocarbon molecules, negation ($\neg$) is applied to the predicate $Connected$. Thus, the rules defining $Connected$ should appear in a stratum strictly before the stratum(s) defining $Inorganic$ or $Hydrocarbon$, which has no impediments. As for the bond fluents on the right hand side of the rules (e.g. $Bond$), note that the rules defining bond fluents, as will be discussed in the next section, do not use any abbreviations on their right hand side. As such, the rules for bond fluents also can always be put in a stratum such that the global stratification is not jeopardized.

Thanks to stratification of the abbreviations, there is always an unique minimal model satisfying these abbreviations. It is known that data complexity of stratified Datalog$^\neg$ is $P - complete$ [24]; therefore queries about these abbreviations can be answered in polynomial time. These abbreviations belonging to Datalog$^\neg$ is in particular an interesting observation, since it means these abbreviations can be expressed in PDDL as well, making them amenable to be used in state-of-the-art planners. In fact, this is the ground for our experiments using PDDL planners in the Experiments Chapter.

However, a key question is whether all abbreviations in organic chemistry comply to the stratified Datalog$^\neg$ requirements or are there exceptions? If there are exceptions, are there workarounds? This is the main topic that we pursue next.

Although vast majority of the chemical classes and functional groups can be expressed as Datalog$^\neg$ rules, it is not the case for all. Recall alkyls: chemical compounds that consist solely of acyclic single bonded carbon and hydrogen atoms such that all atoms have saturated bonds except for one carbon atom that bridges to another functional group. Contrary to the other functional groups that we had encountered before, alkyls do not have a fixed number of atoms. A generic abbreviation defining alkyls need to be such that it can correctly identify the whole chain, no matter how long the chain is, and make sure this chain is not a ring. As such, the key carbon of alkyl, needs to be the carbon atom that bridges to other functional groups via a single bond, because that it the guaranteed carbon atom in any alkyl. The key carbon of some alkyls have been displayed with the key atoms circled in Figure 3.1.

To define alkyls as abbreviations, we need to be able to identify arbitrary backbones connected to the key carbon of the alkyls. A backbone is a chain of carbon atoms (and hydrogen atoms, which are usually implicitly assumed), that ends with a $-CH_3$. An arbitrary backbone can be defined recursively, traversing unidirectional from one carbon atom to another until we reach the terminating $-CH_3$. However, along the walk on the chain, we need to keep track of the carbon atoms visited, and each time traverse to a carbon that has not been visited yet to exclude loops. Otherwise, in case of molecules with rings of carbon, the rule defining backbone abbreviation may

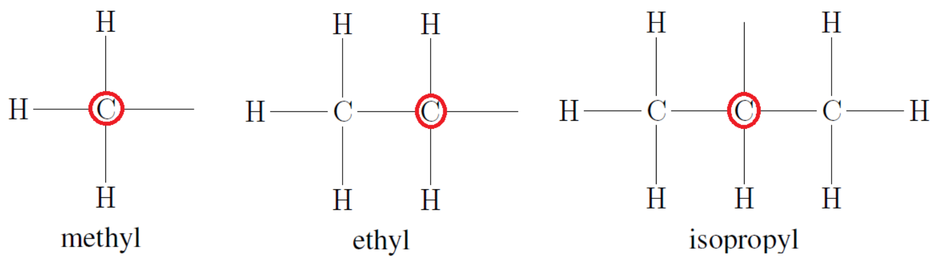


Figure 3.1: Some alkyls with the key atoms circled

cause a non-terminating walk on the ring. This is a key observation, since this is a case (and the only case known in this research) that we require to appeal to a more expressive logic than that of Datalog$^\neg$. More specifically, function symbols are required to check whether a list of carbon atoms of arbitrary length (that has been visited during the walk for identifying backbones) has duplicates or not. Of course, Datalog rules are function-free, and as such the generic abbreviation for backbone, and as a consequence the recursive abbreviation for alkyl with lists (i.e. with functions) cannot be formulated in Datalog$^\neg$.

However, fortunately there are workarounds to represent real alkyls without use of function symbols. Note that it is not the case that arbitrary long rings of carbon atoms are found in the nature. In fact, rings of carbons in monosaccharides rarely exceed 7 or 8, while polysaccharides consist of several short rings bound together. This provides an opportunity to express the backbone abbreviation without function symbols. Of course, this will come at the cost of not being able to identify arbitrary long rings of carbon, but from a practical point of view, arbitrary long ring of carbon do not exist in nature. With this knowledge, two approaches can be used to re-write the abbreviation for backbone without having to use function symbols.

One approach would be to avoid writing recursive rules for backbone. Since we consider a bound on the length of the carbon chains, we can describe each chain of carbon without need of recursion. In this approach, a backbone can be a chain of carbon of length one, or a carbon chain of length two, up to the carbon chain of length say 10, which is the upper bound for chains of carbons in nature. For each helper abbreviation defining a fixed-length chain of carbons, when traversing through the chain, each carbon visited can be simply checked against the previous traversed carbons to ensure pairwise inequality of the carbons. We call this approach the *non-recursive* definition for alkyls. The main problem with this approach is that it is labour intensive, since there is a very large number of isomeric alkyls, and implementing each is time consuming.

The other approach is to define alkyls recursively, but introduce a 10-ary helping predicate that ensures pairwise inequality of the last 10 visited carbon atoms during the walk to identify backbones, very much like how the predicate UNA worked in the definition of phenyl group. This approach avoids function symbols for arbitrary long chain of carbons, since it puts a bound on the ring length, and uses the helper predicate to make sure all the visited carbons during the walk are unique. Since the ring of carbons in nature does not exceed from the arguments of the helper predicate, this approach correctly identifies all the backbones in nature. We call this approach for representing alkyls the *bounded recursive* definition of alkyls.

Either of the two approaches transform the rules defining alkyls such that they are also expressible in stratified Datalog$^\neg$ by getting rid of the function symbol. In other words, with these workarounds, there will be no known chemical concept in organic chemistry that is not expressible as abbreviations in stratified Datalog$^\neg$. As such, all abbreviation can benefit from computational advantages that this logical class provides, including data complexity of $P - complete$, as well as expressibility in PDDL.

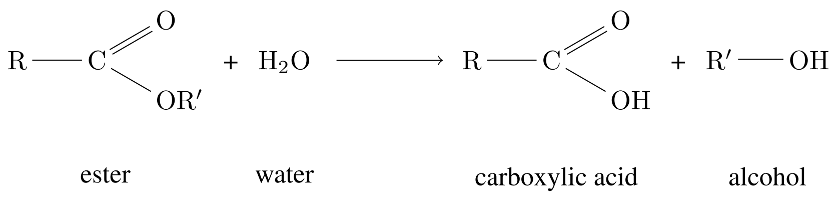
3.2 Representing Reactions

Previously, a few formalisms for representing chemical reactions have been developed, which are based on graph-transformation modeling of chemical reactions. Fujita [31, 32] introduced Imaginary Transition Structure (ITS) to model chemical reactions. ITS is based on the idea that a chemical reaction can compactly be represented by identifying the bonds that cleave, the bonds that form, and the bonds that are not affected. Thus, ITS coins the terminology of *in-bonds* for the bonds existing only in the products of a reaction, *out-bonds* for the bonds that only exist in the reactants and disappear after the reaction, and *par-bonds* for the bonds that are both on the reactants and the products. ITS is in spirit similar to "superimposed reaction graphs", a formalism for modeling chemical reactions developed by G.E. Vladutz [96, 95]. In superimposed reaction graphs, the atoms that do not participate in bond alterations (the atoms with no adjacent in-bonds or out-bonds) are omitted to produce a sub-graph called "superimposed reaction skeleton graph" which focuses on the alterations that a chemical reaction make.

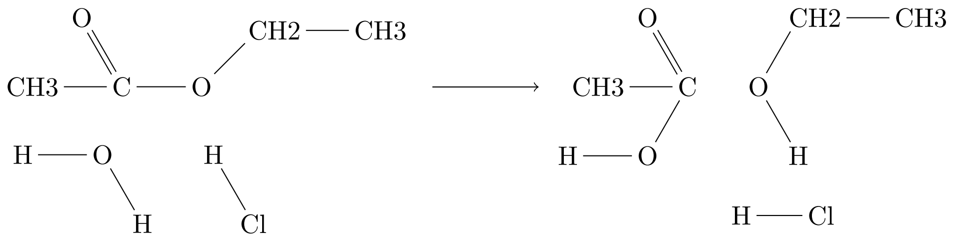
Our representation of chemical reactions is also in accordance to the above formalisms. We introduce two different approaches such that in each approach, as a result of a chemical reaction, a number of new bonds are created, a number of bonds are cleaved, and some bonds do not change. One approach, named the micro approach, models a chemical reaction as a sequence of SC actions that each cleave or form a bond. The other approach is the macro approach, where a chemical reaction is represented with a SC action, whose effects modify all the bonds according to the reaction, and preserves the rest. In each approach, PAs are used to encode the knowledge of when an action is possible. Moreover, the effects of the reactions in both approaches are captured through SSAs, which is based on the idea that as a result of an action, some fluents become true, some become false, and some do not change.

Both approaches, tend to model generic chemical reactions, those involving chemical classes as opposed to specific molecules. Of course, nothing prevents us from being able to represent a chemical reaction between specific molecules, but it is more advantageous to model generic reactions, and our approaches are capable. The advantage that generic reactions provide is that a generic reaction can subsume multiple (tens, hundreds, or more) specific reactions. Therefore a generic reaction models a class of specific reactions compactly.

In the next subsections, each approach is extensively discussed using the following running example. Hydrolysis of esters is a generic chemical reaction, where an ester molecule reacts with a water molecule, in presence of a strong acid as a catalyst, to produce a carboxylic acid and an alcohol. In the below scheme for the reaction, the strong acid catalyst has been eliminated to simplify the matters.



An instance of this reaction is the reaction between ethyl acetate $CH_3-COO-CH_2-CH_3$ (an ester) with water, when the strong acid is hydrochloric acid HCl:



Alcohols are sometimes represented as R−OH, where R stands for any substituent that satisfies the conditions defined above for a molecule to be classified as alcohol.

In this generic chemical reaction, 3 bonds cleave, 3 bonds form, and in total 6 atoms are affected. The ITS for this reaction is represented in Figure 3.2, where in-bonds are represented with a circle on the bond, the out-bonds are crossed off with two lines, and the par-bonds use the common notation for bonds.

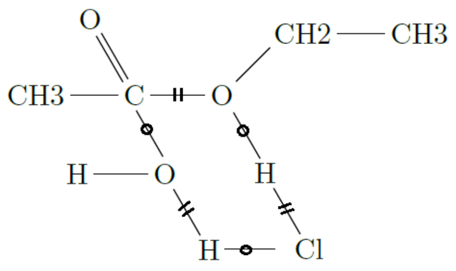


Figure 3.2: ITS for ethyl acetate and water reaction

3.2.1 Micro approach

The micro approach is the lower level approach among the two approaches for representing chemical reactions. In this approach, the most preliminary actions in the process of a chemical reaction are considered as the SC actions: splitting and forming of bonds. In this approach, an action is introduced for creation of each type of bond, and an action is introduced for splitting of each type of bond. Examples are $formBond(C, O)$, which forms a single bond between the atoms C and O, $splitBond(C, O)$, which splits a single bond between the atoms C and O. Similarly, $formDoubleBond(C, O)$ forms a double bond between the atoms C and O, and $splitDoubleBond(C, O)$ splits a double bond between the atoms C and O. Similar actions are introduced for other bond types.

In the micro approach, in addition to the bond fluents, there is a need for some auxiliary fluents to keep track of the key atoms in the process of a chemical reaction, when a reaction has started, but has not ended yet. The need for these auxiliary fluents arises from the fact that in this approach, after each action, only a sole bond will form or cleave, while the reaction is a sequence of such actions. The auxiliary fluents are introduced to identify the intermediate sub-molecules in the midst of a reaction, and enable further continuation of the chemical reaction. These fluents can be created if the mechanism of the reaction is known. Also, since these are fluents,

SSAs are written for them to capture the effect of the actions on them. An example of such SSAs would be:
$HydrogenSeparatedFromStrongAcid(h, do(a, s)) \leftrightarrow$
$\exists x((a = splitBond(h, x) \vee a = splitBond(x, h)) \wedge Strong\_acid(h, x, s)) \vee$
$HydrogenSeparatedFromStrongAcid(h, s) \wedge$
$\neg \exists x(\, a = formBond(h, x) \ \vee a = formBond(x, h)\,)$.
The fluent $HydrogenSeparatedFromStrongAcid(h, s)$ holds if h is a hydrogen ion (a proton) separated from a strong acid in the situation s. This fluent is true in $do(a, s)$ if and only if the most recent action a split the bond of the hydrogen ion from a strong acid, or if the hydrogen ion had been separated from a strong acid in previous situation s, and it did not form a new bond.

Precondition Axioms

As common in BATs, we need to write PAs to express when it is possible to execute an action. In the micro approach, there are actions for splitting and forming of bonds, therefore we need PAs for actions representing splitting and forming of bonds. Each PA will encode all the possibilities where a bond of that type can be cleaved or formed. As such, each PA is long conjunctions of conditions, each of which describing a situation when it is possible that a cleavage or formation happens. To appeal to the running example of hydrolysis of esters, assuming we have only that reaction in our knowledge base, the PA for the action *splitBond* is as follows:
$Poss(splitBond(x,y),s) \leftrightarrow \Pi_{C\text{–}O} \vee \Pi_{H\text{–}O} \vee \Pi_{H\text{–}Cl}$,
where each Π is a condition corresponding to one of the splits that happens as the result of the reaction. $\Pi_{C\text{–}O}$, one of the disjunctive sub-cases, describes the conditions to initiate the reaction by splitting the single bond between the carbon and oxygen in the ester molecule. It stands for:
$Ester(x, y, s) \wedge \exists h', h'', o, x\,(Water(o, h', s) \wedge Strong\_acid(h'', x, s))$.
This condition is expressing all the elements that need to be in place for the reaction to happen: an ester molecule, a water molecule, and a strong acid. Furthermore, it identifies which atoms split the bond: the key atoms in the $Ester$ abbreviation (since x and y as the arguments of the action are also the arguments of the $Ester$ abbreviation). In the abbreviation for ester, the key arguments have been chosen to be the carbon and oxygen atoms that share a single bond with each other (distinguished by arrows in Figure 3.3). The effect of this step of the reaction is displayed in Figure 3.3.

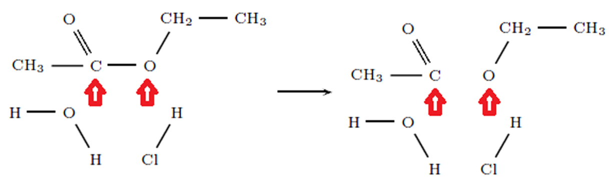


Figure 3.3: *splitBond* between carbon and oxygen atoms

Let us take the next step in this reaction. As mentioned before, for carrying on in the micro approach when a reaction has started, we need some auxiliary fluents. In this case, the auxiliary fluent is $CarbonFromPartialEster(c, s)$ which is true if c has been previously the key atom in an ester that now has split its bond with the neighboring oxygen atom. The condition $\Pi_{H\text{–}O}$ uses this auxiliary fluent to make another step in the reaction, by splitting the single bond between the oxygen and hydrogen atoms in the water molecule. $\Pi_{H\text{–}O}$ is as follows:

$Water(x, y, s) \wedge (\exists c)CarbonFromPartialEster(c, s)$.

This condition is stating that it is possible to split the single bond between atoms x and y, if they are the key atoms of the $Water$ abbreviation (defined to be the oxygen atom in a water molecule and a neighboring hydrogen atom), and there is a carbon atom that was previously part of an ester molecule. In the situation where we have already executed the first step of the hydrolysis of esters, the fluent $CarbonFromPartialEster(c, s)$ is true for some carbon atom, and therefore the reaction can take the next step to split the water molecule. This step of the reaction is represented in Figure 3.4, with the arrows pointing at the carbon from a partial ester and the atoms that split the bond.

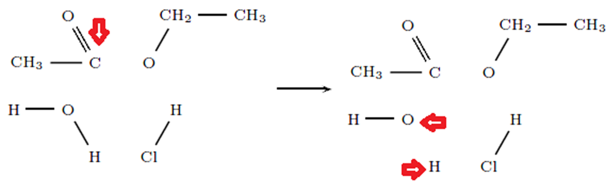


Figure 3.4: *splitBond* between hydrogen and oxygen of water

The third condition $\Pi_{H\text{–}Cl}$ that splits the bond between hydrogen and chlorine atoms in the hydrochloric acid is represented similarly. The PAs for actions representing bond formations is not different from the splitting actions in spirit. Additionally, if we increase the number of reactions in the knowledge base, the corresponding conditions for each bond cleavage or formation will append the conjunctions on the right hand side of the PA for the respective bond.

Successor State Axioms

Not surprisingly, as customary in BATs, we need to write SSAs to account for the effect of each action that is introduced on the fluents of the domain. In the micro approach, each action directly forms or splits a certain bond. Therefore the SSAs for the fluents in this approach are straight forward. If the action formed the bond, the corresponding fluent becomes true. Otherwise, it remains true unless an action splits it. Consider the SSA for the $Bond$ fluent below:

$$\begin{aligned}&Bond(x, y, do(a, s)) \leftrightarrow (a = formBond(x, y) \vee a = formBond(y, x))\\ &\qquad \vee(Bond(x, y, s)\wedge\\ &\qquad\qquad a \neq splitBond(x, y) \wedge a \neq splitBond(y, x)).\end{aligned}$$

The SSAs for the other type of bonds are very similar to the one mentioned above. It is worth noting that in the micro approach, the SSAs for the bond fluents are context-free, i.e. the value of them can be determined only by considering the actions that have been taken. However, this is not the case for the SSAs written for the auxiliary fluents. Considering the example of $HydrogenSeparatedFromStrongAcid(h, s)$, the $Strong\_acid$ is the context condition that needs to be in place for the action to make the fluent true. As such, in general the SSAs for the micro approach are context-dependent.

3.2.2 Macro approach

In contrast to the micro approach, the macro approach is a high level approach that models an entire chemical reaction using a SC action. Therefore, for each (generic) reaction that is included in the knowledge base, a new action term needs to be introduced. The arguments of the actions correspond to the atoms that change bonds in the process of the chemical reaction, or in ITS terms, the atoms that are adjacent to at least one in-bond or out-bond. To appeal to the running example of hydrolysis of ester molecules, this reaction can be represented using the following term, with long name of variables suggesting which atom in the reaction it refers to:

$$hydrolysisOfEsters(carbOfEster_1, oxOfEster_2, oxOfWater_3,$$
$$hydOfWater_4, hydOfAcid_5, acidAnion_6)$$

In the macro approach, there is no need to introduce auxiliary fluents, as the action will carry out all the effects of the reaction at once, and there are no intermediate molecules produced in the process.

Precondition Axioms

The PAs in the macro approach express the conditions that need to be in place for the reaction to happen, as well as give significance to the arguments of the action term. In other words, it is in the PAs that each argument of the action term gets mapped to the corresponding atom in the reaction. Consider the PA for the hydrolysis of esters in the macro approach below:

$$Poss(\ HydrolysisOfEsters(carbOfEster, oxOfEster, oxOfWater,$$
$$hydOfWater, hydOfAcid, acidAnion), s\ ) \leftrightarrow$$
$$Ester(carbOfEster, oxOfEster, s) \wedge$$
$$Water(oxOfWater, hydOfWater, s) \wedge$$
$$Strong\_acid(hydOfAcid, acidAnion, s).$$

This PA is stating that the reaction $HydrolysisOfEsters$ is possible if there is an ester molecule, a water molecule, and a strong acid. Moreover, it asserts that the first two arguments of the action should be the key atoms in the $Ester$ abbreviation, the third and forth arguments should be the key atoms in the $Water$ argument, and the last two arguments should be the key arguments in the $Strong\_acid$ abbreviation.

Successor State Axioms

In the macro approach, each (generic) chemical reaction is represented with a SC action. Having many action terms means longer SSAs, since we need to take into account the effect of all of the actions on the bond fluents. Consider the running example of the hydrolysis of esters once again:

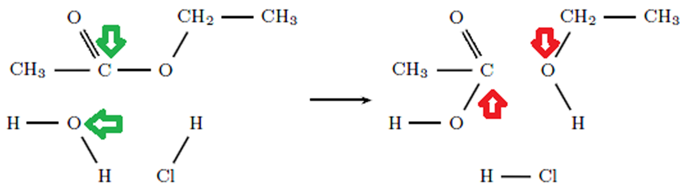


Figure 3.5: Hydrolysis of ethyl acetate. The arrows point to some of the atoms that form or split bonds.

Assuming $HydrolysisOfEsters$ is the only (re)action in our knowledge base, The SSA for the $Bond$ fluent in the macro approach is as follows:

$$
\begin{aligned}
&Bond(x, y, do(a, s)) \leftrightarrow \\
&\quad \exists\, t_1, t_2, t_3, t_4(a = HydrolysisOfEsters(x, t_1, y, t_2, t_3, t_4)) \lor \\
&\quad \exists\, t_1, t_2, t_3, t_4(a = HydrolysisOfEsters(y, t_1, x, t_2, t_3, t_4)) \lor \\
&\quad \exists\, t_1, t_2, t_3, t_4(a = HydrolysisOfEsters(t_1, x, t_2, t_3, y, t_4)) \lor \\
&\quad \exists\, t_1, t_2, t_3, t_4(a = HydrolysisOfEsters(t_1, y, t_2, t_3, x, t_4)) \lor \\
&\quad \exists\, t_1, t_2, t_3, t_4(a = HydrolysisOfEsters(t_1, t_2, t_3, x, t_4, y)) \lor \\
&\quad \exists\, t_1, t_2, t_3, t_4(a = HydrolysisOfEsters(t_1, t_2, t_3, y, t_4, x)) \lor \\
&\quad Bond(x, y, s) \land \neg \\
&\quad (\exists t_1, t_2, t_3, t_4(a = HydrolysisOfEsters(x, y, t_1, t_2, t_3, t_4)) \lor \\
&\quad \exists t_1, t_2, t_3, t_4(a = HydrolysisOfEsters(y, x, t_1, t_2, t_3, t_4)) \lor \\
&\quad \exists t_1, t_2, t_3, t_4(a = HydrolysisOfEsters(t_1, t_2, x, y, t_3, t_4)) \lor \\
&\quad \exists t_1, t_2, t_3, t_4(a = HydrolysisOfEsters(t_1, t_2, y, x, t_3, t_4)) \lor \\
&\quad \exists t_1, t_2, t_3, t_4(a = HydrolysisOfEsters(t_1, t_2, t_3, t_4, x, y)) \lor \\
&\quad \exists t_1, t_2, t_3, t_4(a = HydrolysisOfEsters(t_1, t_2, t_3, t_4, y, x)))\,.
\end{aligned}
$$

As customary in the SSAs, this SSA is composed of two parts: a part describing what actions make the fluent true, and a part asserting what actions should be avoided so that the fluent remains true (if it was already true). In the above SSA, the first seven lines of the right hand side correspond to the first part of the SSA, and the rest correspond to the second part of the SSA. To understand the logic of the SSA, it is important to know what atom each argument of the action is referring to. So recall the action:

$$
\begin{aligned}
&hydrolysisOfEsters(carbOfEster_1, oxOfEster_2, oxOfWater_3, \\
&\qquad hydOfWater_4, hydOfAcid_5, acidAnion_6)
\end{aligned}
$$

All of the actions that can cause two certain atoms x and y to form a single bond are listed with the disjunction symbol ($\lor$) in the first part of the successor state axiom, meaning if the last action was any of these, then the two atoms have a single bond. Notice that after executing $HydrolysisOfEsters$, the key carbon of the ester forms a single bond with the oxygen of water. These two atoms are distinguished with green arrows on the left hand side of the Figure 3.5. The first argument in the $HydrolysisOfEsters$ action corresponds to the carbon of ester, and the third argument refers to the oxygen of water. Therefore, if the last action was $HydrolysisOfEsters(x, t_1, y, t_2, t_3, t_4)$, then x and y have a single bond in between them in the successor state. Similarly, the rest of the actions which can cause two atoms x and y to form a single bond are listed.

However, as a result of this reaction, certain bonds between atoms split. These effects are displayed in the second half of the successor state axiom. The second part of the SSA translates to the following: atoms x and y have a single bond with each other if they already had a single bond and the last action did not cause them to split. In our example of $HydrolysisOfEsters$, after the reaction, the key carbon of the ester splits its bond with the key oxygen of the ester. These atoms are distinguished with red arrows on the right hand side of the Figure 3.5. Note that the first and second arguments of the $HydrolysisOfEsters$ action correspond to the carbon and oxygen of the ester, respectively. Therefore, the atoms x and y have a single bond, only if they had a single bond and the last action had not been $HydrolysisOfEsters(x, y, t_1, t_2, t_3, t_4)$. Thus, there should be no variables t_1, t_2, t_3 and t_4 such that the last action be $HydrolysisOfEsters(x, y, t_1, t_2, t_3, t_4)$. The rest of the actions that cause two atoms to split their bonds are similarly listed.

Had there been more reactions in the knowledge base, their positive and negative effects would have been similarly added to the respective part in the SSA. Here we showed only the SSA for the $Bond$ fluent. The SSAs for the other bond fluents are written in the same vein. It is also worth noting that the SSAs in the macro approach are all context-free, in contradiction with SSAs in the micro approach.

3.3 Discussion

The higher level representation of actions in macro approach makes this approach more appropriate for solving planning problems corresponding to organic synthesis in comparison to the micro approach. An organic synthesis problem requires considerably less actions to be solved if the macro representation is used. It is well-known that the length of a plan is proportional to the time it is required to find it, and as such plans in the macro approach are found much faster than plans in micro approach for a fixed organic synthesis problem. Also, a plan in the macro approach is more meaningful for a chemist since it maps each chemical reaction to an action, while a plan in the micro approach is a sequence of bond split and formations, not providing much information about what chemical reaction has caused the effects.

The low-level representation of chemical reactions in micro approach however, has some advantages the macro approach does not have. For example, the micro approach makes it possible to solve computational problems in chemistry other than synthesis problems. An example of such problems is *mechanism elucidation*. Mechanism elucidation roughly corresponds to determining the internal mechanism of a chemical reaction. It is different from the organic synthesis problem, because it cannot be understood as searching for a sequence of known chemical reactions. In contrast, mechanism elucidation requires finding a sequence of elementary bond splitting/forming operations that constitute a given reaction (informally, process structure of a single reaction has to be determined). It turns out that by utilizing a new set of PAs it is easy to adapt our micro approach to solve mechanism elucidation problems. Instead of formulating preconditions about intermediate molecules – knowledge about them is unavailable when mechanism is unknown – the PAs in the micro approach should allow execution of the elementary actions (bond formation or cleavage) under conditions that respect the rules of organic chemistry. For example, as long as constraints on valences of atoms (and other chemistry constraints based on the first principles) are not violated, the elementary actions should be possible. Thereby, the mechanism elucidation problem is reduced to the AI planning problem, where operators are splitting and forming bonds between atoms, the initial state is characterized by compounds before the reaction starts, and the goal state is a set of molecules produced by the reaction. Using this reduction, we implemented and solved the mechanism elucidation problem mentioned as an example in Section 3.5 of [93].

It is also worth noting some of the key differences in how molecules and reactions are represented in this work and previous works on the matter. A common representation for molecules in the industry is use of connection tables such as MDL Molfiles. A Molfile is a connection table essentially composed of two matrices, one corresponding to the list of atoms in the molecules with their associated properties (type, charge, etc.), and the other a matrix describing which atoms are bonded together and the properties of the bonds. A common reaction representation based on Molfiles is the MDL RXN format and its closely related MDL RD format. A RXN format simply represents a reaction by specifying the substrates and the products of the reaction in Molfile format as a package. It simply includes a section devoted to all the substrates of the reaction, and another for representing the products. A RD format is similar to RXN, and just differs with it in that a number of reactions can be included in the same file, and it allows for some extra associated data to be recorded in the files.

The key difference in how molecules and reactions are represented in this work with that of Molfile, RXN and RD is that we employ a logical language as the expression language, as opposed to a connection table. This representation allows us to connect molecules and reactions to developments in logics, and for example reason about molecules and the effects of the reactions on them within the same language.

However, we are not the first to attempt to utilize logical languages for representing molecules. Prior to us, this has been attempted in the work of Magka [67]. However, there are key differences in our representation with that of Magka, which allows us to reason about generic chemical reactions. For representing molecules, Magka appeals to an abstract *molecule* concept which is related to the atoms and bonds in the molecule via relations such as $hasAtom$ and $hasBond$, respectively. In our case however, there is no abstract *molecule* entity, and a molecule is represented precisely how it really is: a set of atoms and bonds between them. This subtle yet key difference allows us to reason about effects of reactions on molecules easily without the need to produce or destroy abstract *molecule* entities. Magka does not attempt to reason about chemical reactions, but even if the task was attempted, it is unclear whether as a result of a chemical reaction, an abstract entity *molecule* is destroyed and new instances of these abstract entities are produced (possibly more than one), or it is preserved and transformed, and some other *molecules* are created to account for the departing parts. Overall, the extra abstract entity of *molecule* adds a lot of complexity when it comes to reasoning about reactions, as the reactions in reality do not effect these concepts, rather simply affect the bonds between atoms.

It is also worth noting that to address knowledge acquisition in our approach, it is possible to write tools transforming the formats commonly used in the industry to the representation used in this work. In fact, we have developed programs for this purpose. The connection tables include all the information necessary for representing a molecule in our format, and thus there are no obstacles for their translation. As for chemical reactions, our representation requires a reaction to be atom mapped before translation, meaning each atom in the substrates of a reaction have a corresponding atom in the product of the reaction. Fortunately, there are available software capable of performing atom mapping on reaction file formats such as RXN or RD (not limited to these formats), such as ChemAxon's Standardizer [15]. When a reaction is balanced and atom-mapped, all the necessities are in place to translate the reaction to the axioms in our representation that represent it.

Chapter 4

Organic Synthesis Planner

The main contribution of this thesis is to formulate organic synthesis problem as Artificial Intelligence planning problem, and develop a planner capable of solving organic synthesis problems. The planner developed in this work, called the *ChemPlanner*, uses the SC to formulate organic synthesis problems as AI planning tasks. The previous chapter discussed how chemical reactions and molecules can be represented in BATs augmented with abbreviations. This chapter focuses on how planning is done using ChemPlanner, given an organic chemistry BAT.

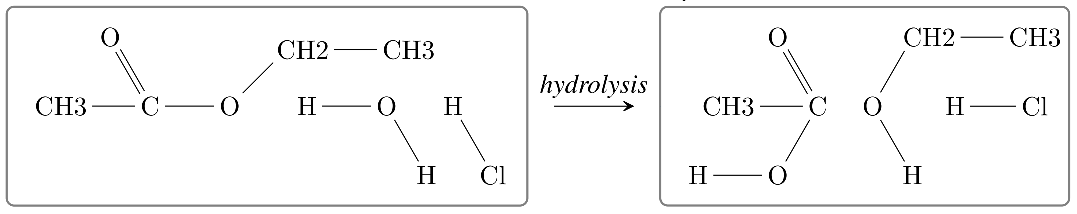
Previously, two approaches for representing chemical reactions were introduced, namely the micro and macro approaches. The ChemPlanner is only compatible with the macro representation. The micro representation, breaks a single chemical reaction into the most elementary actions of bond formation and cleavage, and in a sense loses the structure in the reactions. The ChemPlanner is built on the idea to take advantage of the structures in the chemical reactions, and thus is not compatible with the micro approach. Moreover, a plan for an organic synthesis problem in micro approach is considerably longer than one in the macro approach. It is well-known that the size of the search space is exponential in the length of the plan, and lengthier plans require more time to be discovered. As such, the macro approach solves an instance of an organic chemistry synthesis problem in a shorter time in comparison to the micro approach. Also, a plan generated from the macro representation is more meaningful for a chemist, as it lays out the sequence of chemical reactions necessary to synthesize the goal molecule, as opposed to a sequence of split or form bond operations. Nevertheless, it is possible to solve organic synthesis problems using micro approach as well, but macro approach is more advantageous for this purpose.

The ChemPlanner works with a knowledge base of chemical reactions, represented by the macro approach discussed previously. It accepts a set of initial state molecules, a set of goal molecules, along with a few options for configuration that will be discussed shortly in this chapter. As the output, it produces the sequence of reactions (from the knowledge base) that can transform the initial molecules to the goal molecule, if any. It employs a number of techniques to expedite finding solutions to organic synthesis problems which are either directly borrowed from or inspired by the advancements in the field of state-of-the-art planning. On a high-level, the planner works as follows. Given a concrete problem, it creates an abstract problem, the solution of which will be refined to derive the solution to the concrete problem. Solving the abstract problem can still be challenging due to the number of available actions and the size of initial state and goal molecules. As such, delete list relaxation heuristic is used in the abstract state-space to guide the search. In addition to the delete list relaxation heuristic, the search in the abstract state-space is guided with domain dependent heuristics, inspired by causal graphs, but not exactly complying to them. In the subsequent sections, we will describe how each stage of the planning in ChemPlanner is done in more details. First, how the problems are abstracted. Second, how the solutions to the abstract problems are refined to concrete problems. Lastly, how the abstract plans are actually found.

4.1 Problem Abstraction

The concrete problem in organic chemistry is composed of a set of initial and target molecules, described in the form of their constituent atoms and the bonds between them. In realistic synthesis problems, the number of chemical atoms as the objects of the domain easily exceeds 50, which in turn makes the state-space huge. Interestingly, the atoms in a molecule are not directly associated to the chemical reactivity of the molecule, rather the chemical reactivity is due to groups of atoms in the molecule, namely the constituent functional groups. This lays the foundation for the abstraction performed in the ChemPlanner. The idea is to ignore the details of what atoms and bonds are present in a situation, and focus on what functional groups, chemical classes, and chemical conditions exist instead, as abstract entities. With this knowledge at hand, it is possible to predict on a high-level, which chemical reactions are possible, and what products, in terms of functional groups and chemical classes will be produced. We refer to functional groups, chemical classes and chemical conditions collectively as *high-level molecules* from now on.

A concrete state-space in organic chemistry is a directed graph where the vertices represent states, composed of the atoms and bonds between them, and the edges represent chemical reactions applicable in each state transforming it to a new state. Given an organic chemistry state-space, the ChemPlanner aims to define the abstraction mapping φ to map a state s in the concrete state-space S to an abstract state s' which represents the high-level molecules in s as abstract entities. The edges of the abstract state-space AS correspond to abstract (re)actions, derived from concrete reactions as follows: they require the high-level molecules in their corresponding concrete reaction to be applicable, and produce and destroy all the high-level molecules that the concrete reaction produces or destroys. The cost of all reactions, regardless of abstract or concrete is assumed 1, and therefore are not represented at all. For example, consider the partial concrete state-space where the edge represents the familiar hydrolysis of esters reaction, and the nodes represent the states before and after the reaction. The node on the left is the state where there is an ethyl acetate, a water and a molecule of hydrochloric acid. The node on the right represents the state where there is an acetic acid, an ethanol and a hydrochloric acid molecule.



The corresponding partial abstract state-space is:

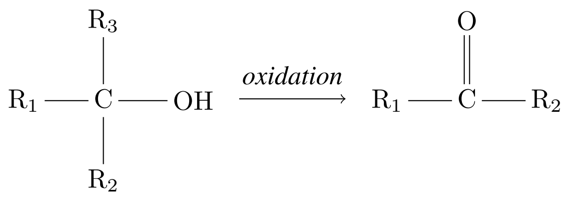
$$\begin{tikzcd}\boxed{\textit{ester water strong-acid carbonyl} \ldots} \arrow[r, "\textit{abstract}", "\textit{hydrolysis}"'] & \boxed{\textit{carboxylic-acid alcohol strong-acid carbonyl} \ldots}\end{tikzcd}$$

In this example, the *abstract hydrolysis* is the abstract version of the familiar hydrolysis of esters: the preconditions for this reaction are abstract entities *ester*, *water*, and *strong-acid*. The positive effects of this reaction are abstract entities *carboxylic-acid* and *alcohol*, and the negative effects are abstract entities *ester* and *water*. If the abstract (re)actions produce and destroy all the high-level molecules that their concrete counterparts do, it is clear that φ defines a proper abstraction for the concrete state-space. Any edge $s \xrightarrow{a} s'$ in the concrete state-space S has its counterpart in the abstract state-space AS, meaning $\varphi(s) \xrightarrow{a'} \varphi(s')$ is in AS, where a' is the abstraction of a. Moreover, if a state s is defined as the goal state in S, meaning $s \in G_S$, then $\varphi(s) \in G_{AS}$ as well. Since there are no excessive edges defined for the abstract state-space to invalidate the homomorphism requirement, φ defines a proper abstraction for the concrete state-space S.

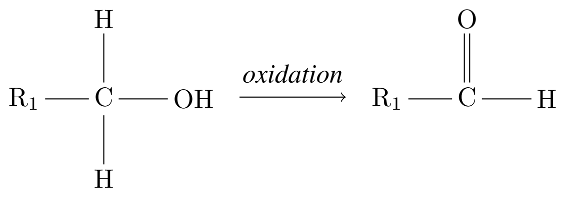
As such, problem abstraction is done as follows in the ChemPlanner. The knowledge base of chemical reactions is preprocessed offline once to derive the abstract knowledge base of chemical reactions, the process of which will be discussed in subsection 4.1.1. Then, given a concrete organic synthesis problem, the initial and goal molecules are processed to extract the high-level molecules therein. The high-level molecules in the initial and goal states form the abstract initial and goal states in the abstract state-space. Subsequently, a new planning task is formulated, asking to find the sequence of abstract actions that can transform the set of high-level molecules in the abstract initial state, to a state in which all the high-level molecules of the abstract goal state exist. The solutions to this new planning task will be refined to derive the solutions to the concrete problem, the process of which will be discussed in section 4.2. The ChemPlanner works with a fixed set of predefined high-level molecules: all the high-level molecules that are in the preconditions of the knowledge base of chemical reactions in the ChemPlanner.

4.1.1 Preprocessing: Computing High-Level Effect of Reactions

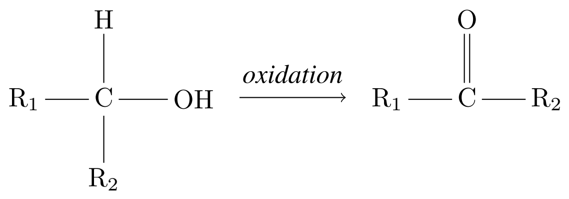
The abstract (re)actions are of vital importance, since correctness of the abstraction relies on them doing their job correctly. The effects of the abstract (re)actions should cover all the high-level effects of their concrete counterpart, otherwise the upward solution property of the abstraction is lost. To demonstrate, consider an organic synthesis problem which has a plan of length one, that of oxidation of alcohols. For the sake of simplicity, assume that oxidation of alcohols is the only reaction available in the knowledge base. The oxidation of alcohols is a well-studied chemical reaction, in which the single bond between the carbon of the alcohol and the hydroxyl group is transformed to a double bond between the carbon atom and oxygen. The reaction needs some chemical conditions as well, which are not central to this discussion, and as such are abstracted away. Consider the generic scheme of oxidation of alcohols below, where R_1, R_2 and R_3 are arbitrary sub-molecules, with the constraint that not all three of them can be carbon atoms, since tertiary alcohols resist oxidation:



The oxidation of primary alcohols typically produces an aldehyde:

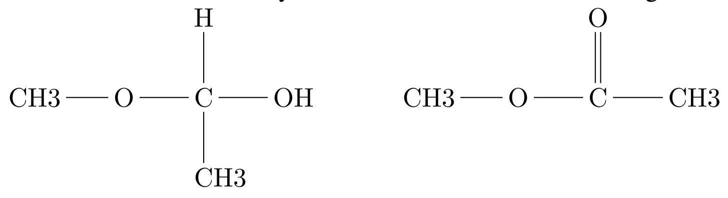


The generic oxidation of secondary alcohols is displayed below, which results in a ketone:



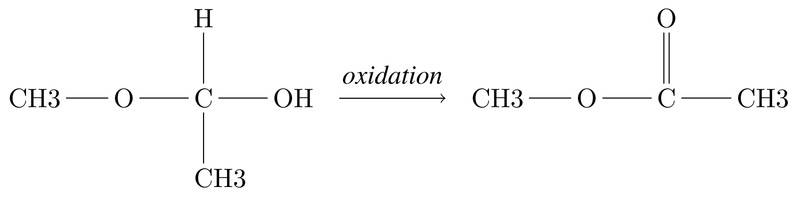
As noticeable from the generic scheme of above reactions, the guaranteed effect of the oxidation of alcohol reaction is the double bond formation between oxygen and carbon atoms, resulting in a carbonyl group. With the perspective of abstract entities, the aldehyde and ketone groups that are formed are by-products of the reaction: they are created because the carbonyl group was created. Nevertheless, assume the abstraction of oxidation of alcohol is designed such that it has as its positive effects the entities *carbonyl*, *aldehyde* and *ketone*.

Now consider the following organic synthesis problem, where given the initial molecule below, we want to know which sequence of reactions produces the goal molecule. Assume that in the initial state, the minimum chemical conditions necessary for oxidation of alcohol are also given.



Initial state molecule Goal molecule

The oxidation of alcohols solves this concrete problem in one step, complying to the generic scheme of the reaction presented above:



However, the abstract problem has no solution in this example. To elaborate, recall that the problem abstraction is done by extracting the high-level molecules in the initial and goal molecules, which will in turn form the abstract problem's initial and goal states. As such, the abstract initial state in this case would be:

alcohol hydroxyl alkyl . . .

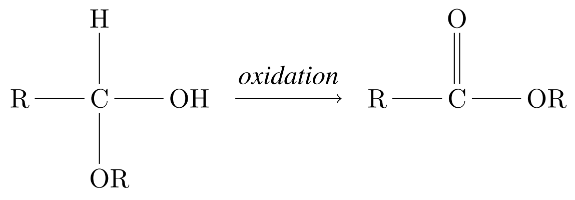
And the abstract goal state would be:

carbonyl alkyl ester . . .

Notice that in the goal state, the abstract entity *ester* exists, but the abstract oxidation reaction does not have it as a positive effect. As such, although the abstract oxidation reaction would be applicable in the abstract initial state, it would not result in a state in which the abstract goal is satisfied. In other words, this problem does not have an abstract solution, while it clearly has a concrete solution. This is because *ester* is a potential high-level by-product of the oxidation reaction, but has been neglected and not included as a positive effect of the abstract reaction. This example demonstrates that if all high-level effects of a reaction are not included in the abstract reaction, the upward solution property of the abstraction is lost, and the planner might not find a proper plan for the problem, even if one exists.

The preprocessing procedure in ChemPlanner, responsible for providing the knowledge base of abstract reactions is designed specifically to ensure the high-level by-products of a chemical reaction are included in the abstract reaction. It proposes a systematic methodology for doing so. The main idea is in order to guarantee comprehensive inclusion of potential by-products of a concrete reaction, all known types of the participants of the reaction are examined to derive what high-level molecules they can result to. In order to identify all the known types of a given participant in a reaction, existing chemical ontologies can be used. ChemPlanner uses the ChEBI ontology for this purpose.

In the ideal case, the preprocessing procedure addresses the problem presented earlier with the example of oxidation of alcohols, when there were no plan of length one in the abstract state-space. The concrete oxidation of alcohols has alcohol as one of its preconditions. Therefore, during the preprocessing, the effects of the reaction on all children of alcohol will be computed and included in the abstract reaction. One of the children of alcohol in ChEBI ontology is *hemiacetal*. Hemiacetal is a primary alcohol with the generic formula $R_1R_2C(OH)OR$, where R_1 or R_2 is often hydrogen and R (bonded to O) is not hydrogen. The effect of oxidation on hemiacetal will result in an ester molecule. As such, the abstract entity *ester* will be included as the positive effect of the abstract oxidation reaction. In turn, this fixes the problem that we encountered before, and the example we presented earlier will have a solution in the abstract space, as well as the corresponding solution in the concrete space. The effect of oxidation reaction on hemiacetal resulting in an ester is displayed below:



hemiacetal ester

The methodology for computing the high-level effects for the reactions assumes a perfect ontology, such that no child of a high-level molecule is missing, and all the children have their structures available. Unfortunately, this is not the case in reality. There is no guarantee that the ChEBI ontology provides all the children of a chemical class. Furthermore, on occasion, the structures of some entries in the ChEBI ontology are missing. This our algorithm from computing the high-level effects of the reaction on that specific child, and potentially valuable high-level effects are lost. However, it is noteworthy that these are the limitations of the ontology, and not the limitations of the methodology for computing the high-level effects intrinsically.

It is noteworthy that the preprocessing phase provides a sort of domain informed causal dependency on the abstract notions of high-level molecules. By appealing to an ontology such as ChEBI for determining the children of a molecule, it accesses domain-dependent information not given in the structure of the problem. By computing the effects of the reactions in terms of abstract high-level molecules, it creates a type of causal relation between them, such that a specific abstract entity like *ester* will implicitly find a causal relation over another abstract entity, like *alcohol* through the abstract reaction, for example *abstract hydrolysis of esters* reaction. This causal relation does not comply to the classical causal graphs used in planning, but is similar to it in spirit. Similar to causal graphs which was founded on the idea of using the causal structure between state variables, the causal relation creates a causal structure on the abstract entities. Of course, in the case of ChemPlanner's causal relation, an abstract entity cannot by itself cause another, as it requires the other abstract entities that make the abstract reaction possible as well.

4.2 Abstract Plan Refinement

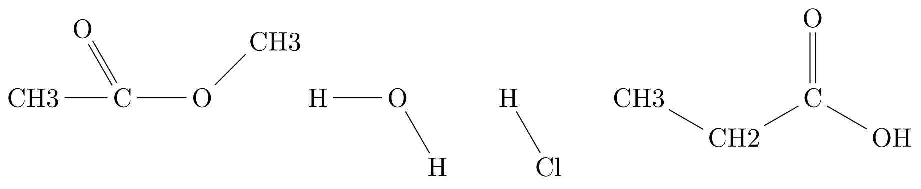
The refinement strategy for ChemPlanner is straight forward and simple. Given an abstract plan, composed of a sequence of abstract reactions, the corresponding sequence of concrete reactions are considered as a skeleton to find the concrete plan. Recall that each reaction in the macro approach was of the form $A(\vec{x})$, where A was the reaction name, and $\vec{x}$ was the list of atoms that change bonds in the reaction. There is a one to one correspondence between action names in the concrete and abstract state-spaces, as such determining the name of the concrete action names is easy from the abstract action names. However, the abstract plan knows nothing about the atoms and bonds between them in the concrete state, and as such the arguments of the concrete actions cannot be produced from the abstract plan. In the concrete actions produced for refining the abstract plan, the arguments of each action are initialized with distinct variables. The main task of the refinement is to find a correct substitution for the variables that make the plan satisfy the goal statement.

In more details, after generating the sequence of concrete actions $A_1(\vec{x}), A_2(\vec{y}) \ldots, A_n(\vec{z})$ with variable arguments, the refinement works as follows. It first considers the very first action in the sequence $A_1(\vec{x})$. It then consults the PA for this action in the initial situation S_0 and tries to satisfy it by finding a substitution $\vec{x'}$ for the variables $\vec{x}$ that make the grounded action possible in the initial situation S_0. It then considers the next action in the sequence $A_2(\vec{y})$, and consults the PA for this reaction in situation $do(A(\vec{x'}), S_0)$ to find a substitution $\vec{y'}$ for variables $\vec{y}$ that makes the action $A_2(\vec{y'})$ possible in situation where $A_1(\vec{x'})$ has been executed. This procedure is continued until all actions in the sequence have found a proper substitution. Next, the resultant situation $s = do(A_n(\vec{z'}), \ldots, do(A_2(\vec{y'}), do(A_1(\vec{x'}, S_0))) \ldots)$ is checked against the goal statement to verify if it satisfies the requirements. If it does, the concrete plan has been found and it is s. Otherwise, if s does not satisfy the goal, the actions in the sequence are revisited in reverse order to find other proper substitutions, and each time the resultant situation is checked against the goal statement. That is, first, preserving all the substitutions for actions A_1 to A_{n-1}, all the possible substitutions for $A_n(\vec{z})$ are considered. Then, in case none of the resultant final situations satisfied the goal, preserving the substitutions for actions A_1 to A_{n-2}, the combination of all the possible substitutions for $A_{n-1}(\vec{u})$ along with $A_n(\vec{z})$ are tried. This procedure is continued until all possible substitutions for all actions and combinations of them are exhausted. If this exhaustive search does not find a situation that can satisfy the goal, it means that the abstract plan cannot be refined to a concrete plan. The procedure is the same in the case that before finding a substitution for all the actions for the first time, an action $A_i(\vec{t})$ in the sequence cannot find a proper substitution. In this case also, all the actions before $A_i(\vec{t})$ in the sequence are revisited in reverse order to find new substitutions, and each time the resultant situation is used to test whether a proper substitution for the action $A_i(\vec{t})$ can be found.

Now that the abstraction and refinement strategies in ChemPlanner have been discussed, we can discuss some of their properties. Upward solution property is guaranteed in our abstraction, provided that the abstract reactions add and remove all the high-level effects of the concrete reactions. That is, if there is a concrete plan, there also exists an abstract plan. To prove, for the sake of contradiction, assume that there exists a concrete plan, but no abstract plan. This is only possible if at least one of the high-level effects of the concrete reactions in the concrete plan is not included in the effects of the corresponding abstract reactions. But this contradicts the assumption that the abstract reactions add and remove all the high-level effects of the reactions.

Moreover, the abstraction and refinement strategies in ChemPlanner satisfy the ordered monotonicity property in an informal sense, but not formally. The ordered monotonicity property informally requires a concrete plan to be derivable from an abstract plan while preserving its general structure. As discussed before, each concrete plan in ChemPlanner is derived from the abstract plan as the skeleton, and as such preserves its structure. In the formal sense however, for the ordered monotonicity property to be granted, the steps in the concrete plan should be used intact in the concrete plan. This is not the case in ChemPlanner's refinement strategy, as discussed before, due to a different nature of abstraction used in ChemPlanner.

The ChemPlanner's abstraction however, does not have the downward refinement property. That is, not every solution to the abstract problem can be refined to a concrete plan. To demonstrate this, we use a simple example and for the sake of simplicity, assume that the only reaction in the knowledge base is the hydrolysis of esters. The given initial state is composed of a molecule of methyl acetate, a molecule of water and a hydrochloric acid molecule. The goal is a propanoic acid. These are displayed below:



methyl acetate water hydrochloric acid propanoic acid

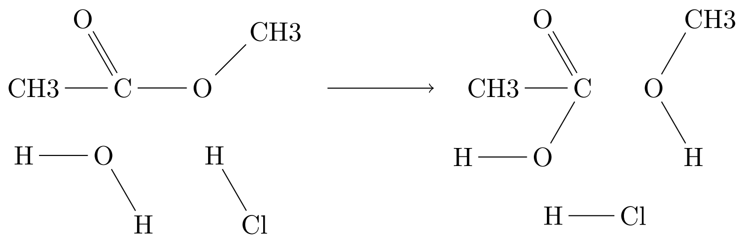
The corresponding abstract problem created for this problem has the following initial state:

ester water strong-acid ...

while the abstract goal state is:

carboxylic-acid alkyl ...

Clearly, the abstract reaction hydrolysis solves the abstract problem, since it has the *carboxylic-acid* in its positive effects, as one would expect. However, if the concrete hydrolysis of esters is applied on the concrete initial state, acetic acid will be created and not propanoic acid.



In fact, the propanoic acid cannot be produced with the aforementioned starting molecules, yet the abstract problem had a plan. The abstract plan had found a way to produce a carboxylic acid, which is the chemical class of the goal molecule, but the concrete problem requires a different carboxylic acid than what can be produced. In any case, this shows that the downward refinement property does not hold for our abstractions.

4.3 Finding Abstract Plans

Once the abstract problem corresponding to a given concrete synthesis problem has been formulated, it is time to find the abstract plans. Abstract plans intuitively produce guesses of sequence of reactions that can potentially solve the synthesis problem. Starting from the high-level molecules in the initial state, an abstract plan finds the sequence of reactions that if executed sequentially result in a situation in which the abstract goal is satisfied. As such, it is a planning problem on its own, where finding the sequence of abstract reactions is the goal.

Solely abstracting the concrete problem is not enough to reduce the search space to a magnitude that planning can be done with satisfactory speed. It is well-known that the size of search space to be explored in the worst case is exponential in the length of the plan, and lengthy plans require a lot of search. Another factor that affects the search space by affecting the branching factor is how many objects, in this case high-level molecules are present. In general, the more high-level molecules, the greater the search space. Therefore, to achieve satisfactory results, the search in the abstract state-space is guided with heuristics to expedite finding the abstract plans.

ChemPlanner uses a variation of A^* search guided by delete list relaxation heuristics for finding the plans in the abstract state-space. Two delete list relaxation heuristics are supported by ChemPlanner: the additive heuristic h_{add} and max heuristic h_{max}. A^* search is a well-known best-first search algorithm that explores the nodes with least heuristic values first. The heuristic function in A^* is a sum of two functions $g(s)$ and $h(s)$. In A^* search, $g(s)$ is the known cost of the state s from the initial state, and $h(s)$ is the estimated cost of s to a goal state. In ChemPlanner, since each action by default has cost of one, $g(s)$ stands for the number of actions taken from the initial state to reach to state s, that is, the length of situation resulting in s. The $h(s)$ on the other hand is computed from the h_{add} or h_{max}, the choice of which depends on the user. ChemPlanner follows the standard A^* search with one exception. In the case that the $h(s)$ for a situation s is estimated zero, the cost for $g(s)$ is excluded from computing the overall heuristic for the situation. The rationale for this is that, when $h(s)$ is zero, it means that for sure s satisfies the goal condition. As such, there is no point in delaying processing it, as it can be considered a solution to the abstract problem right away and passed on for refinement.

Algorithm 1 describes the algorithm for finding the abstract plans. In this algorithm, the user decides which of the h_{add} or h_{max} should be used to compute $h(s)$. Moreover, the user specifies a bound B on the length of the plans considered during search. The bound B makes sure that search will always terminate by making each branch in the search tree finite. However, due to this bound, search may terminate prematurely, i.e. without reaching a goal state. Therefore, to be useful, the bound B should overestimate the length of a plan. The priority queue Q in Algorithm 1 is used to keep visited nodes in the search tree, commonly known as the frontier, where the nodes are sorted in ascending order based on their heuristic value. Any situation that is extracted from the priority queue is first checked to see whether it can satisfy the goal. If not, if the length of the situation has reached the maximum bound specified by the user, it will not be expanded. Otherwise, the children of the situation will be identified, heuristic values for them will be computed, and each will be inserted in the queue Q. This process in continued until an abstract plan is found, or the search space defined by the bound B is exhausted.

Algorithm 1 Finding Abstract Plans

```
//the input to the algorithm is the abstract initial state and abstract goal state,
//along with the choice of which heuristic should be used to compute $h$, and the bound $B$
//the output is a plan for the abstract problem
initialize by putting $S_0$ corresponding to the abstract initial situation in a priority queue $Q$
while $Q$ is not empty do
    extract the first element in the $Q$ namely $Current$
    if $Current$ satisfies the abstract goal then
        return $Current$
    end if
    if $length(Current) < B$ then
        identify all applicable actions in $Current$, namely $Actions$
        for each action $act$ in $Actions$ do
            compute the delete list heuristic $h(s)$ for the situation $s$ resulting from executing $act$ in $Current$
            if $h(s) \neq 0$ then
                set the heuristic value associated to $s$ by adding $g(s)$ and $h(s)$
            else
                set the heuristic value associated to $s$ to zero
            end if
            insert $s$ in $Q$
        end for
    end if
end while
return null, denoting the search space has been exhausted
```

As discussed before, not every abstract plan can be refined to a concrete plan in ChemPlanner. Therefore, the algorithm for finding abstract plans should make it possible, in case an abstract plan fails in refinement phase, to search for another plan. This necessity is provided in the algorithm by simply carrying on the algorithm from the point that it had halted after finding an abstract plan, and pretending that the situation did not result in an abstract plan. As such, if the length of the situation does not exceed the bound B, it will be expanded and its children will be placed in the queue. This simple step makes it possible to continue the search for other plans without neglecting any situations, guaranteeing complete search within the depth bound. To sum up, the algorithm is capable of finding all the plans bounded by the length B in the abstract problem, if any.

Chapter 5

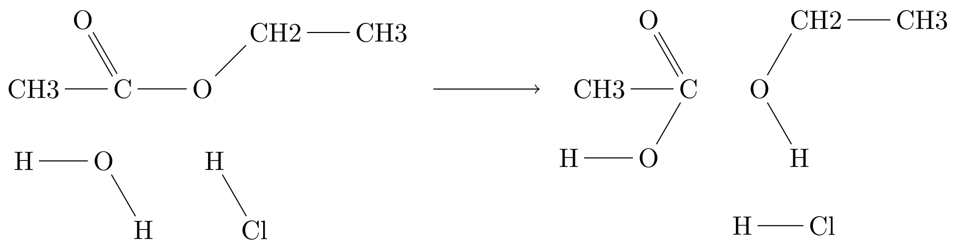
Implementation and Experiments

This chapter includes experiments aiming to elucidate how the micro and macro approaches for representing reactions compare in a planning setting. Moreover, it discusses ChemPlanner's implementation and compares its performance with a number of other planner systems on a set of benchmark problems. Lastly, experimental results obtained from employing ChemPlanner for solving an enriched version of benchmark problems introduced in [41] are presented.

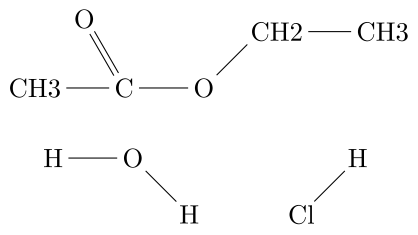
5.1 Micro vs Macro for Planning

This section aims to provide a sense how the performance of the micro and macro approaches for representing reactions differ when they are employed to solve organic synthesis problems. In order to provide a fair comparison ground, the same organic synthesis problem was formulated in both approaches. For the macro approach, the goal molecule could be synthesized by executing one chemical reaction: hydrolysis of esters. As for the micro approach, the plan was constructed from six split and formation bonds. Search for the plans in both approaches was the same: an iterative deepening depth first search.

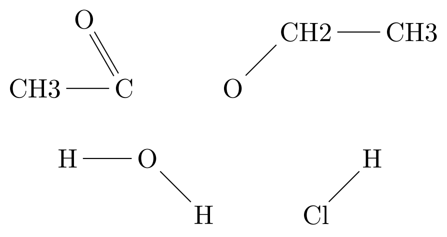
In more details, the organic synthesis problem consisted of an ethyl acetate molecule in the initial state, as well as a water molecule and a hydrochloric acid as a strong acid. The goal was to produce an alcohol. The macro approach achieved the goal by executing one reaction, as displayed below:



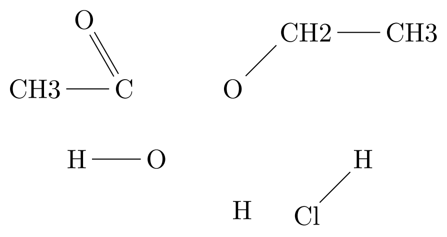
The sequence of six micro actions corresponding to completing the hydrolysis of esters given the initial molecules described above are as follows. The initial state:



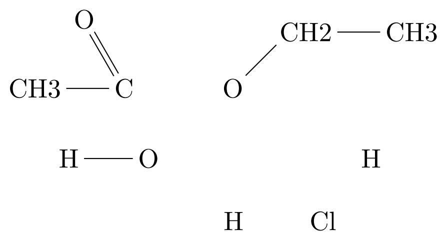
The first micro action splits the bond between the carbon and oxygen in the ethyl acetate.



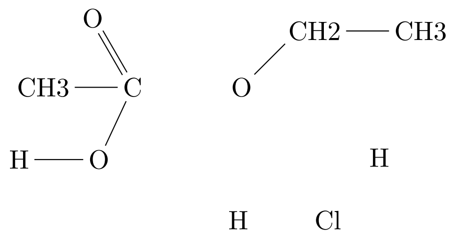
The second micro action splits the bond between the water molecule.



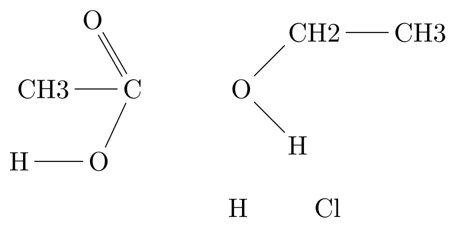
The third action splits the bond between hydrochloric acid.



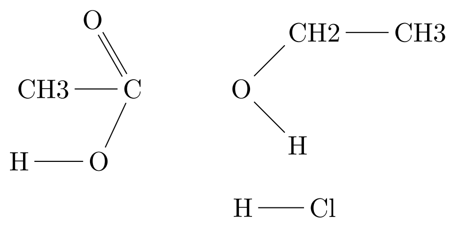
The fourth action forms bond between the carbon atom of the former ester and the oxygen of the former water.



The fifth action forms bond between the hydrogen atom of the former acid and the oxygen of the former ester.



The 6th action forms a bond between the hydrogen of the former water and the remaining of the acid to rebuild the acid catalyst.



The machine that was used for the experiment used a 2.30 GHz Intel(R)Core(TM)i7-3610QM CPU, and an allocated 1GB of RAM for the process, although memory usage for this experiment is insignificant. The macro plan was found in 0.14 seconds, while it took the iterative deepening algorithm 963.62 seconds to find the corresponding sequence of six micro actions. This is not a huge surprise, since it is known that planning time has direct relation with the length of the plan, and that search space grows exponentially in plan length.

The micro approach has an advantage of representing the mechanism and detailed actions that may happen in a chemical reaction. The actions in micro approach are low level and demonstrate hypothetically what may happen in the chemical reaction. But one of the disadvantages of this approach would be a relatively big number of actions required to take for one chemical reaction to complete. In addition, one specific chemical reaction can be represented with more than one sequence of actions and there might be no unique sequence of actions. Finally, when having chain of chemical reactions, in the micro approach it is unclear where the boundaries between the chemical reactions are. All that exists is a sequence of low level actions between atoms (split and forming bonds) and it is unclear what action belongs to what chemical reaction and when one chemical reaction ends and the other begins.

The macro approach has the advantage that in comparison with micro approach, it requires less actions to complete. This should make planning problems faster and easier. Additionally, there is a clear boundary between chemical reactions in a chain of chemical reactions and it is easy to distinguish them from each other. But as for the disadvantages, using this approach we lose the mechanism of representing the lower level of details which actions occur between atoms in a chemical reaction. A chemical reaction will be regarded as a black-box where detailed actions that take place in a chemical reaction are implicit.

5.2 ChemPlanner's Implementation

This section briefly describes ChemPlanner's implementation and the knowledge base of chemical reactions it used for conducting experiments in the rest of this paper, as well as a brief argument about its soundness and completeness.

The knowledge base of chemical reactions in ChemPlanner includes 55 generic chemical reactions, subsuming the knowledge base that Heifets used for conducting experiments in [41], but not exactly complying to it. In more details, the knowledge base used in ChemPlanner is a significantly richer revision of the chemical reactions used in [41], which differ from it in that:

- Hydrogen atoms are represented and reasoned about explicitly.
- The reactions are made balanced and full effects are represented. Note that in [41], most, if not all reactions were unbalanced, and not all the effects of the reactions were represented. This required significant effort since all reactions from [41] had to be manually edited and completed.
- On occasion, a chemical reaction was split into two. This was applied to the reactions in the library in [41] that from chemistry's point of view were commonly regarded as two separate reactions, but appeared as a package in [41] since they commonly follow each other. An example of such reactions are *Grignard Reagent Formation* and *Grignard* reactions. These two commonly follow each other in chemical synthesis problems, and were represented as one reaction in the work of Heifets, but are actually two separate reactions, since *Grignard Reagent Formation* does not necessarily have to be followed up by *Grignard*, although it usually does.

This knowledge base of chemical reactions enables us to employ ChemPlanner to solve the same set of organic synthesis problems that were used in [41]. In fact, section 5.3.2 is dedicated to experimental results obtained from trying the benchmark problems from [41].

ChemPlanner was also provided with 75 abbreviations (corresponding to high-level molecules discussed in Chapter 4) that were used in the ChemPlanner's planning process. These abbreviations coincide with the chemical classes, functional groups and chemical conditions that were used in the preconditions of the knowledge base of chemical reactions. Furthermore, this set included no abbreviation that required recursive definition. Recall that recursive definitions in abbreviations were necessary if the abbreviation required to take into account an unknown number of atoms in the process of validating the abbreviation. Some examples that were presented previously include *inorganicCompounds* or *MoleculeContainingFluorineAndOxygen*. The set of abbreviations in ChemPlanner has no such abbreviations. Moreover, regarding abbreviation for alkyl, recall from Chapter 3 that we discussed two approached for representing it in Datalog$^{\neg}$. ChemPlanner is equipped with the non-recursive implementation of alkyls, where all alkyls containing 4 carbon atoms or less are identified.

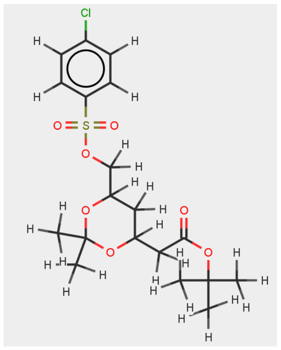
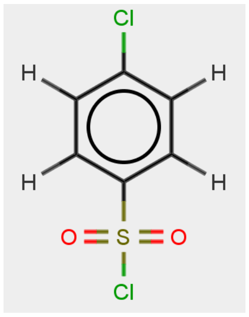
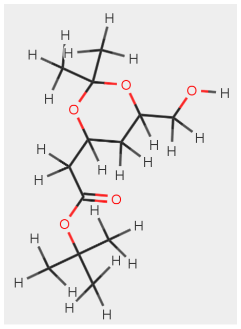
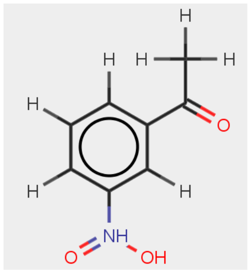
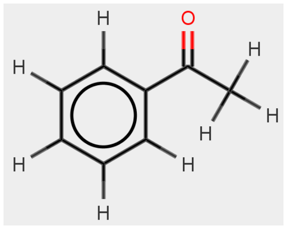
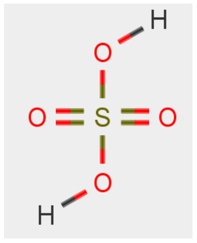
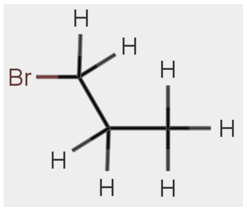
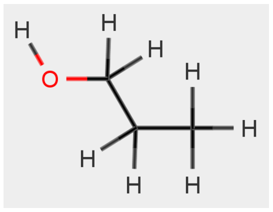
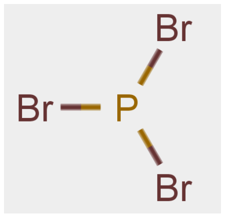
ChemPlanner has been implemented in ECLiPSe PROLOG, and as such the main deduction machinery it uses is SLDNF. However, it is known that there is no guarantee of termination of SLDNF algorithm on recursive rules (see Section 2.4.3). Luckily, due to the set of abbreviations used in ChemPlanner and the fact that they were all non-recursive, ChemPlanner is immune to SLDNF's difficulty when it comes to recursive rules, and thus it is guaranteed completeness and soundness with respect to this set.

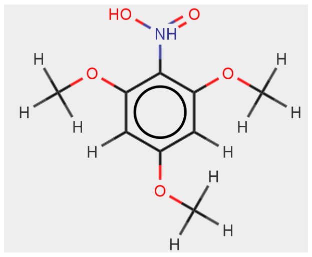
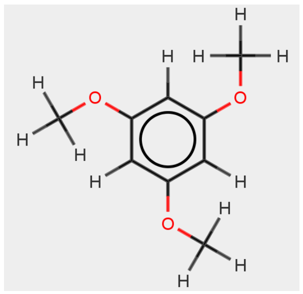
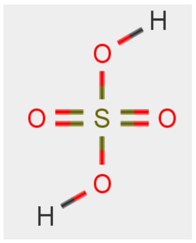
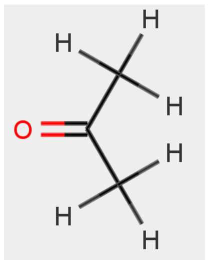
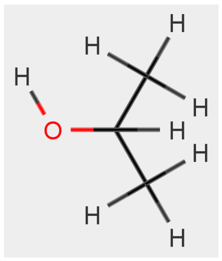
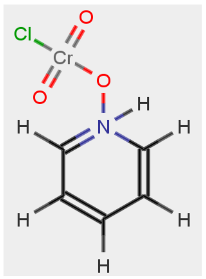
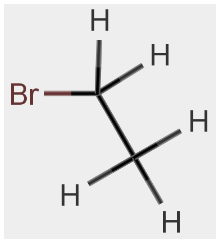
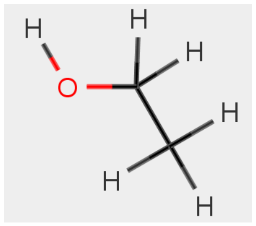
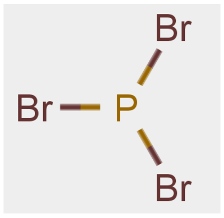
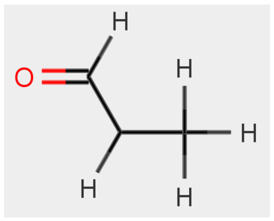
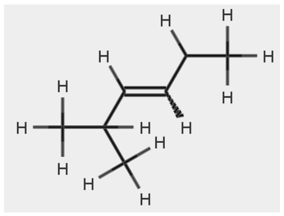
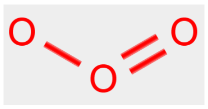
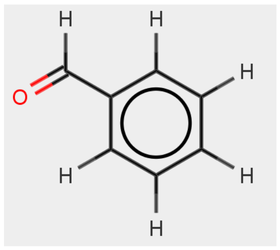
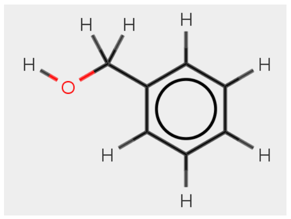
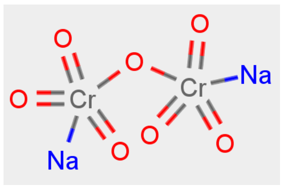
5.3 Experimental Results

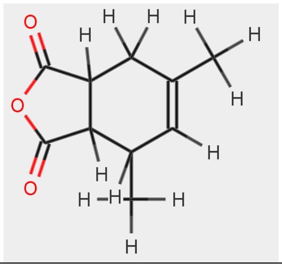
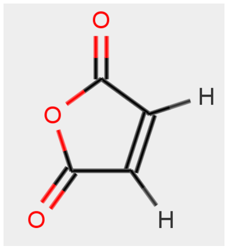
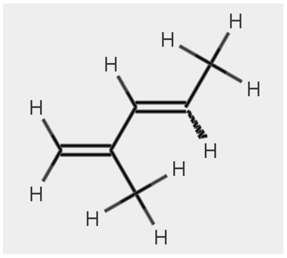
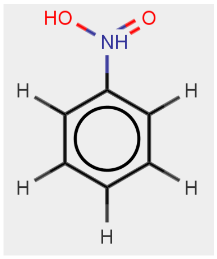
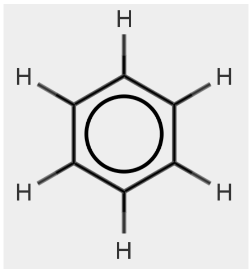
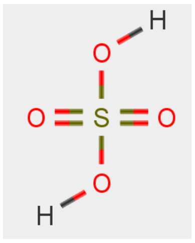
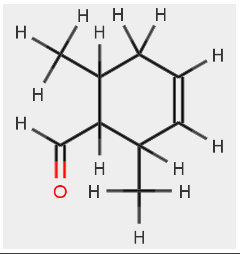
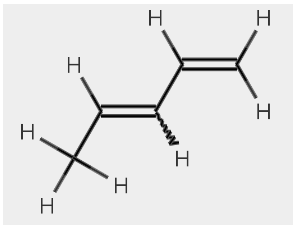
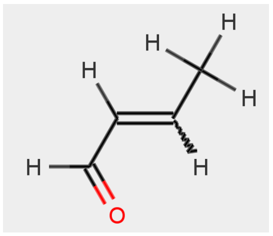
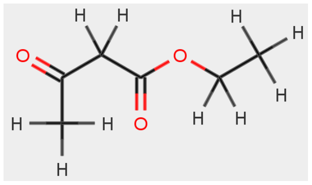
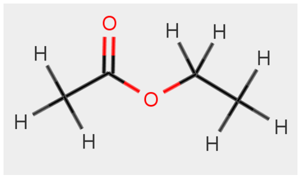
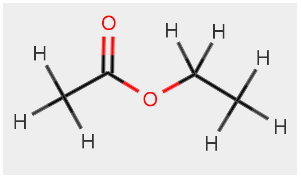
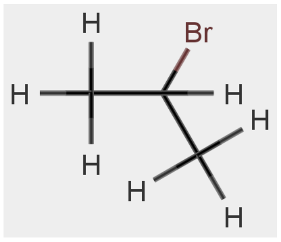
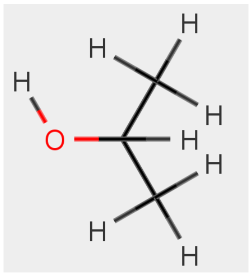
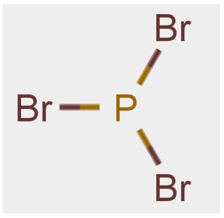
This section is composed of two subsections, each presenting and discussing experimental results obtained from a set of benchmark problems. The first set of benchmark problems was creates by us, while the second one is an enriched version of the problems used in Heifets work [41].

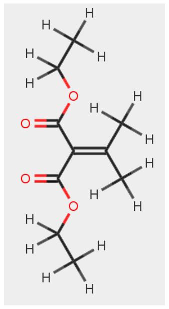
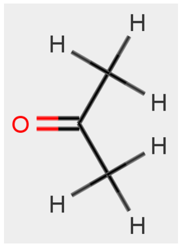
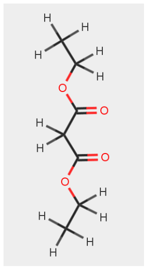
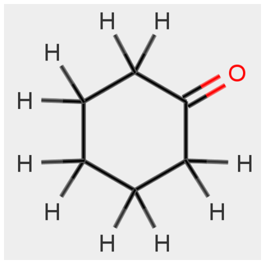
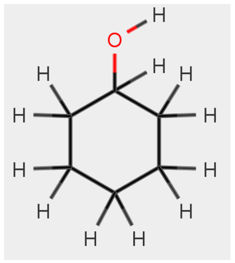
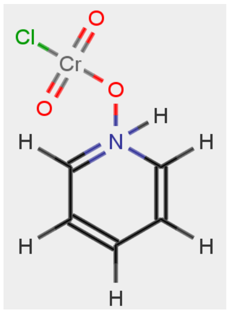
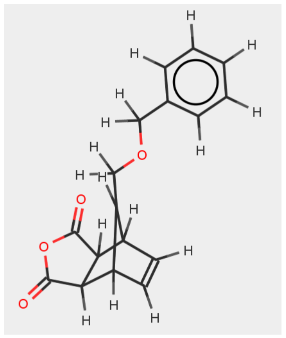
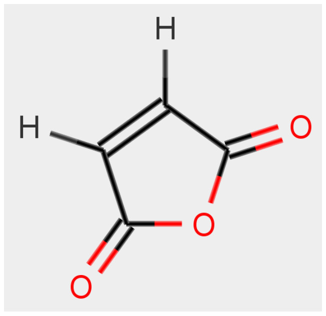
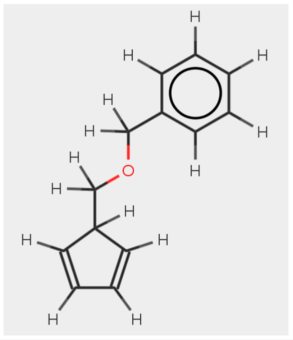
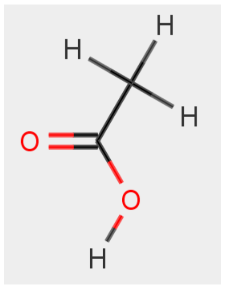
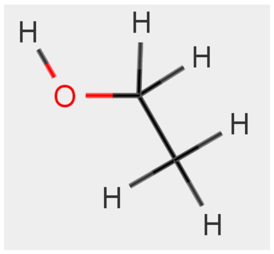
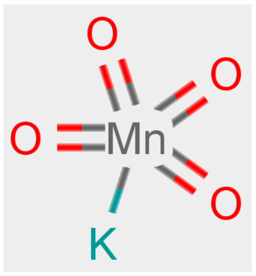
5.3.1 Results for the First Set of Benchmark Problems

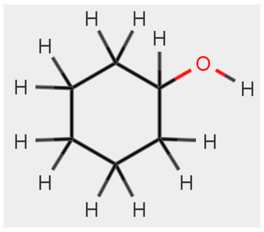
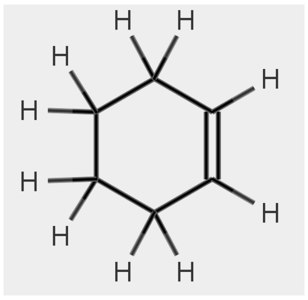
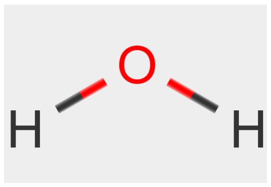
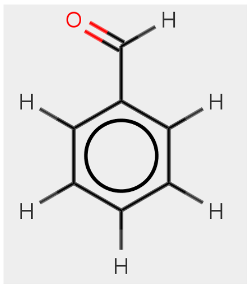
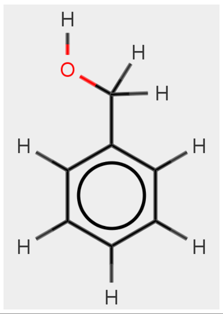
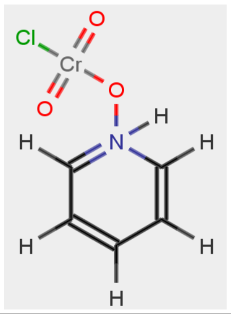
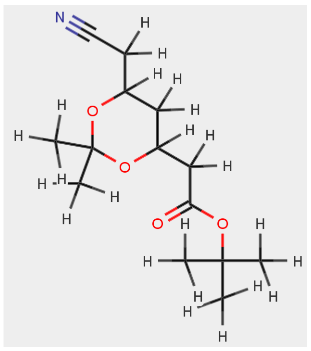
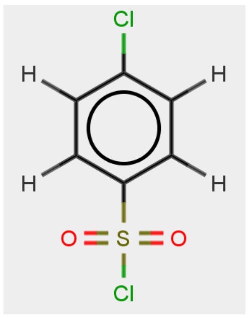
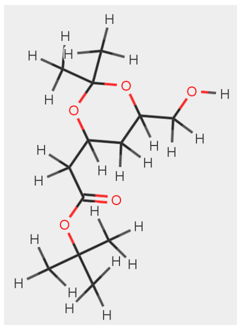
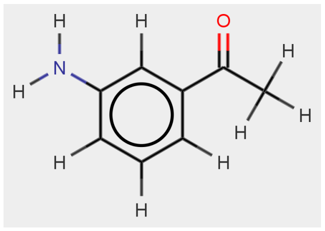
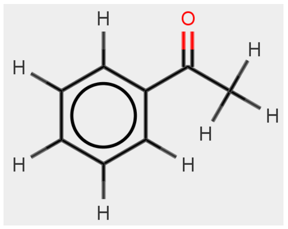
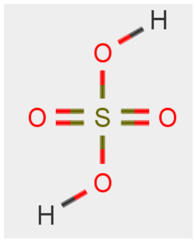
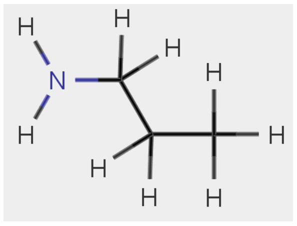
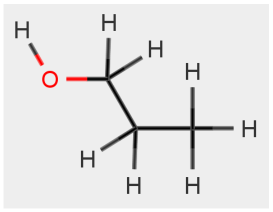
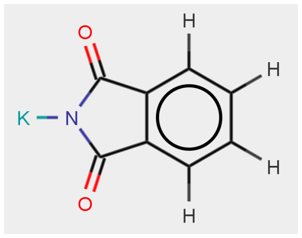
This subsection presents empirical experiments conducted on ChemPlanner and compares the results with results obtained from a number of other planning systems employed to solve the same set of problems. The benchmark problems used in this section include 47 organic synthesis problems of different complexity. Table 5.1 presents the problems in this benchmark where each problem is given a number for identification. The table displays for each problem the goal molecule that its synthesis was attempted, and two of the molecules that existed in the initial state. It is worth noting that in most cases, other than the two molecules displayed in the initial state molecules, other molecules also existed, but they have not been displayed here for lack of space.

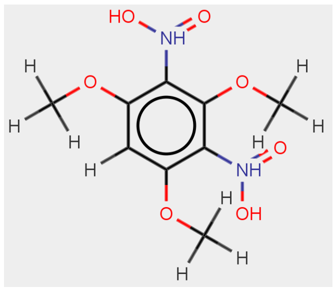
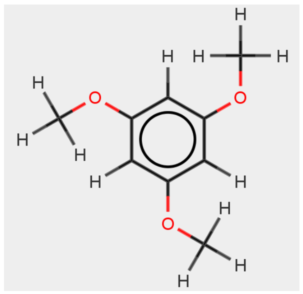
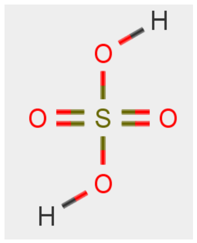
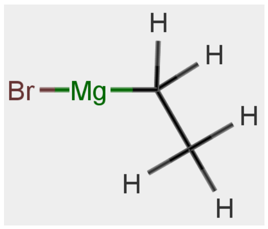
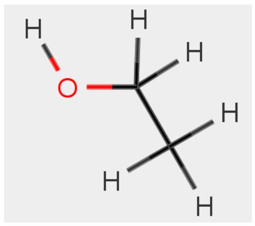
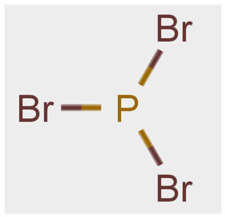
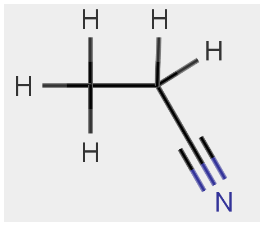
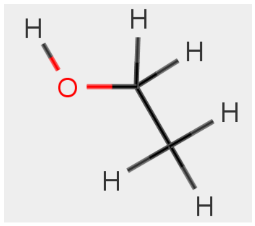
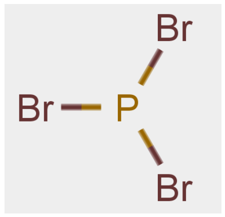
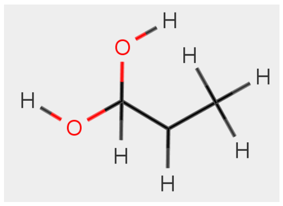
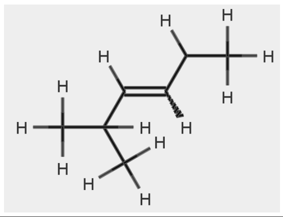
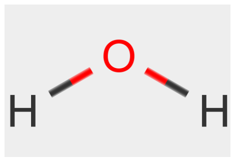
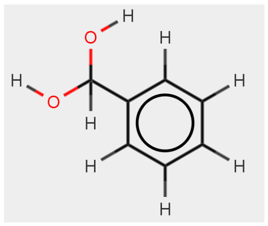
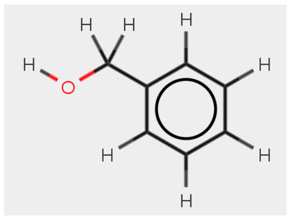
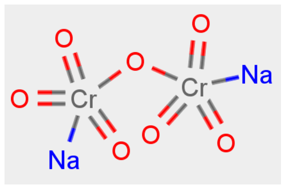
Prob.	Goal Molecule	Init. Molecule 1	Init. Molecule 2
1			
2			
3			

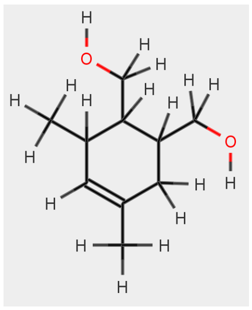
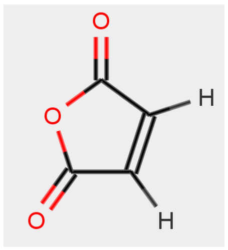
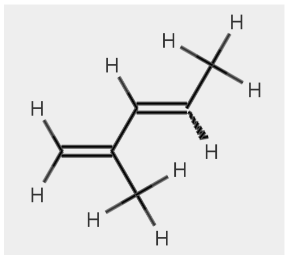
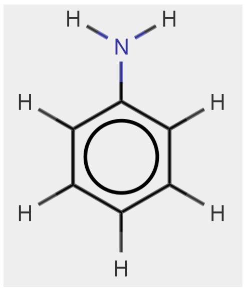
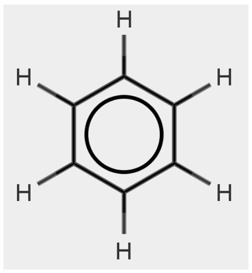
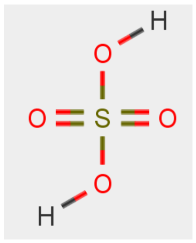
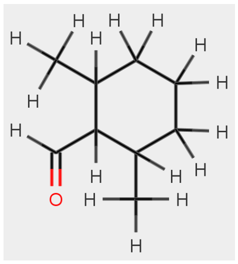
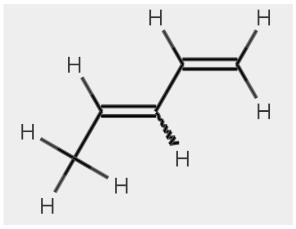
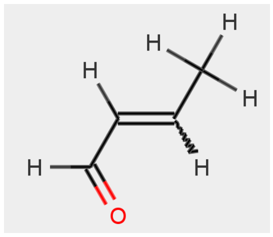
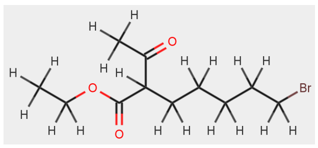
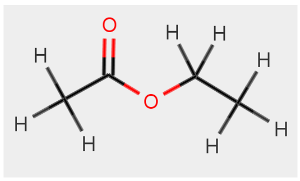
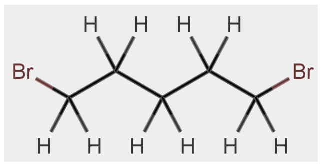
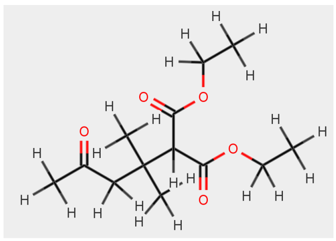
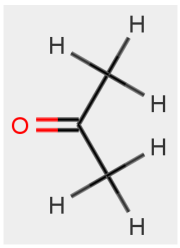
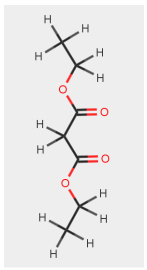
4			
5			
6			
7			
8			

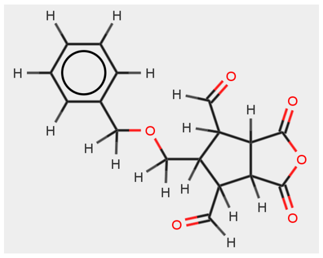
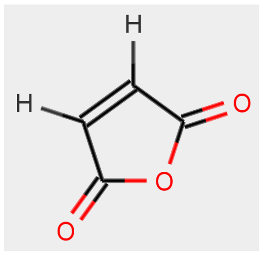
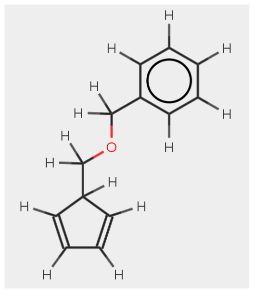
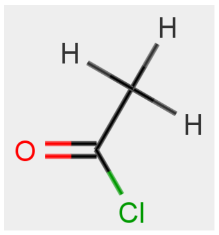
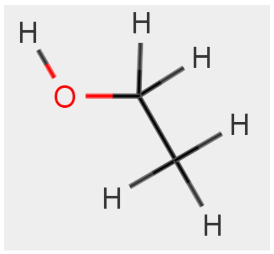
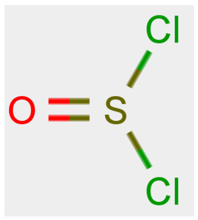
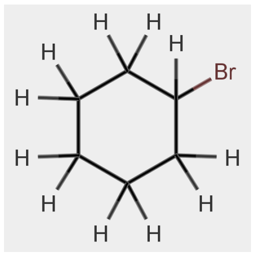
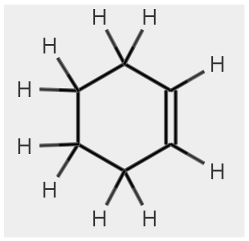
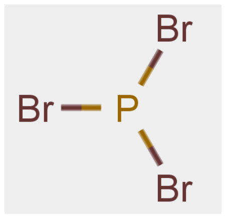
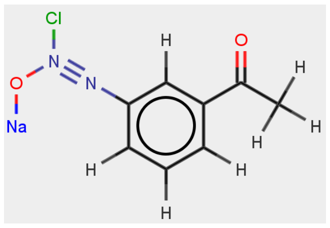
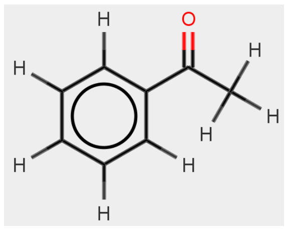
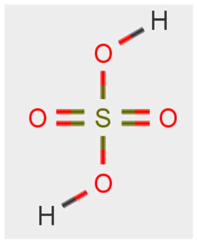
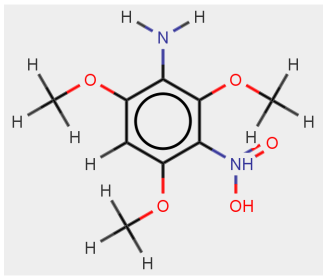
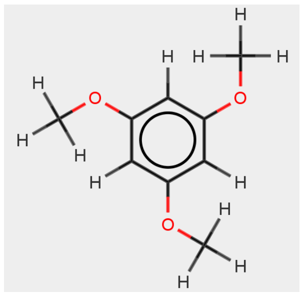
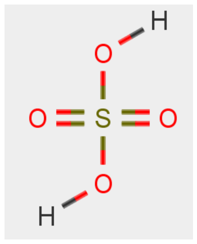
9			
10			
11			
12			
13			

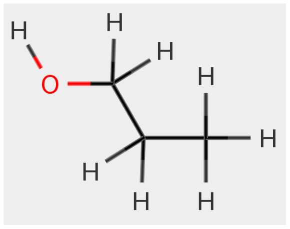
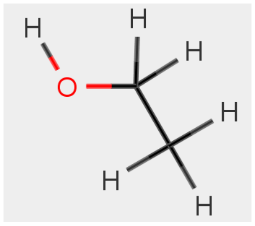
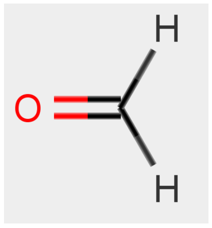
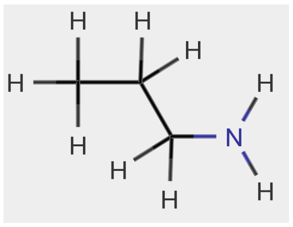
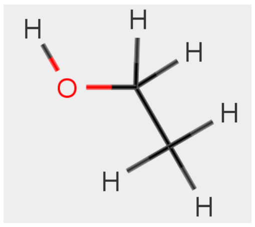
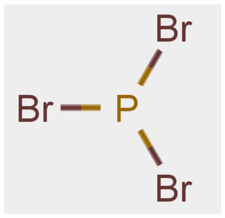
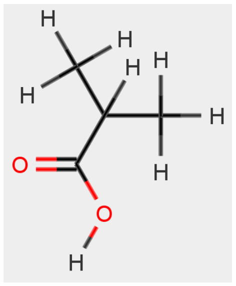
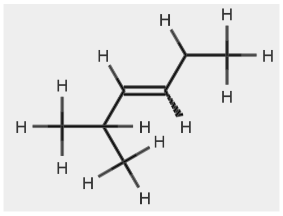
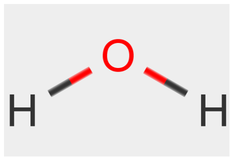
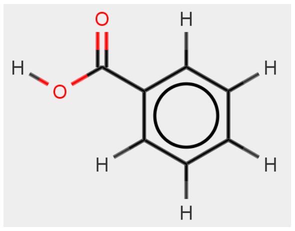
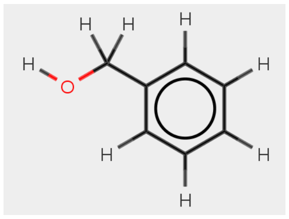
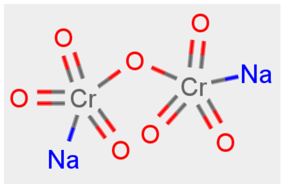
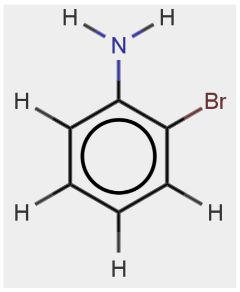
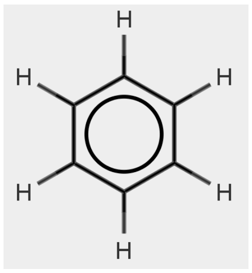
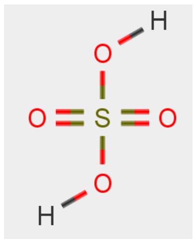
14			
15			
16			
17			

18			
19			
20			
21			
22			

23			
24			
25			
26			
27			

28			
29			
30			
31			
32			

33			
34			
35			
36			
37			

38			
39			
40			
41			
42			

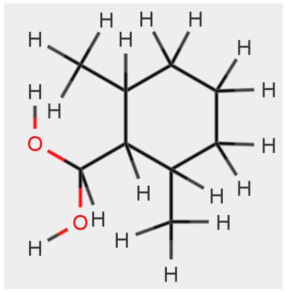
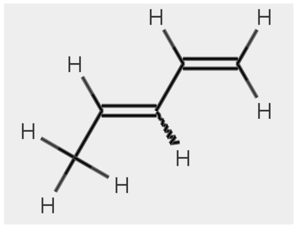
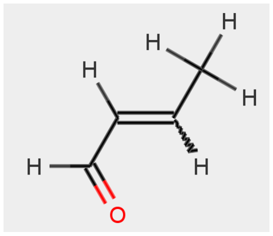
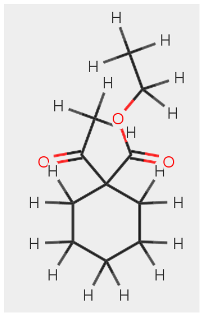
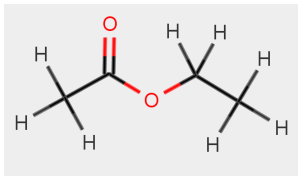
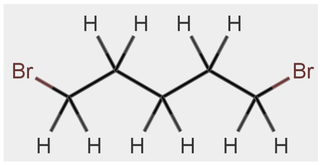
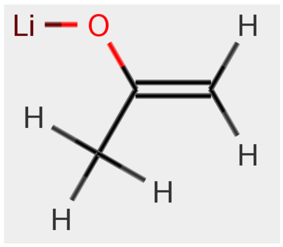
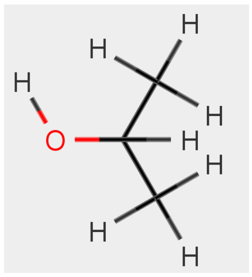
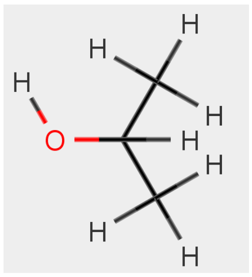
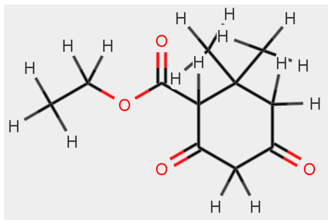
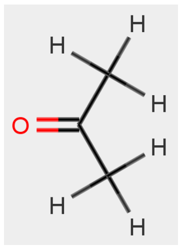
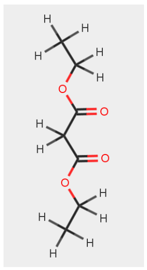
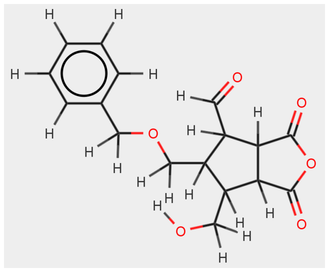
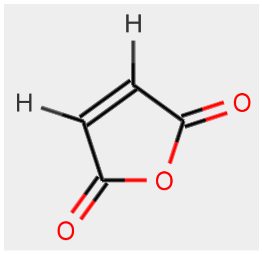
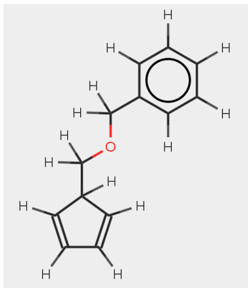
43			
44			
45			
46			
47			

Table 5.1: The benchmark problems

In order to provide ground for comparison, aside from ChemPlanner that was discussed in Chapter 4, a number of other planner systems, employing either variations of the original ChemPlanner's algorithm, or radically different algorithms were used for solving the benchmark problems. A brief description of each algorithm is provided below, which corresponds to the table 5.2 as well.

- **Additive** corresponds to the original ChemPlanner's algorithm when additive heuristic was used.
- **Max** corresponds to the original ChemPlanner's algorithm when max heuristic was used.
- **ChemPlanner ID** is a variation of the original ChemPlanner. It is identical to ChemPlanner except in that, instead of using A^*-like search and delete relaxation heuristics for finding abstract plans, it uses *iterative deepening* (ID) algorithm.
- **ID** is the algorithm were no problem abstraction is performed, and the problem is attempted to be solved in one level using iterative deepening (ID) search algorithm.
- **1L Add** is the algorithm were no problem abstraction is performed, and the problem is attempted to be solved in one level using A^*-like search guided with additive heuristic.
- **1L Max** is the algorithm were no problem abstraction is performed, and the problem is attempted to be solved in one level using A^*-like search guided with max heuristic.

Table 5.2 includes the results of the experiments using each algorithm on the benchmark problems, where the times are reported in seconds. All the experiments have been conducted on a machine with 2.30 GHz Intel(R)Core(TM)i7-3610QM CPU, when maximum of 1GB of RAM had been allocated to the process. Also, a cut off time, set to 2000 seconds had been introduced to limit the execution time. The dashes in the table indicate that the problem did not complete in 2000 seconds.

Prob. #	Plan length	Additive	Max	ChemPlanner ID	ID	1L Add	1L Max
1	1	19.297	19.25	19.485	19.297	9.938	9.454
2	1	2.558	2.48	2.465	2.464	2.683	2.543
3	1	2.418	2.449	2.45	2.434	2.496	2.511
4	1	2.356	2.418	2.465	2.449	2.746	2.496
5	1	2.683	2.637	2.684	2.652	2.87	2.746
6	1	2.324	2.34	2.324	2.589	2.558	2.184
7	1	2.917	2.87	2.995	2.98	3.65	3.448
8	1	2.387	2.355	2.386	2.418	2.667	2.481
9	1	3.447	3.385	3.448	3.339	2.87	2.684
10	1	2.325	2.309	2.371	2.371	2.574	2.371

11	1	3.26	3.307	3.292	3.51	2.839	2.698
12	1	9.672	9.562	9.656	9.703	9.999	9.532
13	1	2.34	2.293	2.434	2.34	2.543	2.433
14	1	4.867	4.727	4.68	4.727	3.962	3.9
15	1	4.742	4.711	4.274	4.259	5.21	5.226
16	1	5.444	5.351	5.304	5.273	4.29	4.274
17	1	2.246	2.294	2.262	2.262	2.589	2.465
18	1	3.073	3.136	3.198	3.385	2.855	2.871
19	1	2.543	2.605	2.574	2.496	2.87	2.683
20	2	36.879	37.753	37.612	582.087	729.43	965.927
21	2	2.605	2.62	2.636	12.448	103.366	106.673
22	2	3.088	3.135	3.12	3.744	9.735	10.094
23	2	2.496	2.434	2.511	3.526	135.003	175.673
24	2	2.324	2.402	2.371	2.418	2.652	4.275
25	2	2.324	2.262	2.402	2.449	2.559	4.29
26	2	3.183	3.213	3.166	4.414	399.955	419.05
27	2	2.402	2.48	2.433	2.715	31.044	30.856
28	2	4.571	4.618	4.586	5.351	105.987	105.862
29	2	2.262	2.309	2.34	2.356	8.626	8.627
30	2	4.103	4.134	4.056	4.306	122.492	122.227
31	2	19.734	19.937	12.574	674.111	-	-
32	2	25.818	25.958	10.28	585.456	-	-
33	2	5.71	6.021	6.287	64.491	871.281	871.796
34	2	2.34	2.262	2.434	2.356	9.687	9.625
35	2	3.167	3.105	3.308	368.911	287.635	454.899
36	3	3.276	3.666	3.385	316.167	-	-
37	3	2.621	2.668	2.558	123.896	-	-
38	3	2.73	2.605	2.496	2.964	3.634	3.697
39	3	2.589	2.574	2.589	3.213	4.711	4.758
40	3	17.877	17.722	24.274	479.173	-	-
41	3	224.455	224.907	441.327	159.402	510.56	503.041
42	3	2.699	2.761	2.481	15.039	676.951	674.595
43	3	5.398	6.973	6.115	188.262	-	-
44	3	84.1	-	-	-	84.88	-
45	3	288.867	289.335	283.11	329.209	-	-
46	3	1723.827	304.872	-	-	-	-
47	3	9.547	17.644	241.645	-	-	-

Table 5.2: Experimental results

Results in Table 5.2 are obtained from two runs of each algorithm and averaging the results. Table 5.3 displays the standard deviation between the two runs for each algorithm.

Prob. #	Additive	Max	ChemPlanner ID	ID	1L Add	1L Max
1	0.154	0.132	0.253	0.154	0.165	0.120
2	0.021	0.055	0.142	0.012	0.011	0.033
3	0.110	0.043	0.120	0.076	0.011	0.120
4	0.110	0.033	0.087	0.044	0.043	0.021
5	0.055	0.131	0.021	0.000	0.098	0.011
6	0.044	0.033	0.033	0.131	0.087	0.154
7	0.021	0.309	0.066	0.000	0.065	0.000
8	0.044	0.221	0.032	0.043	0.000	0.032
9	0.010	0.220	0.021	0.131	0.011	0.043
10	0.076	0.077	0.055	0.033	0.066	0.066
11	0.099	0.055	0.055	0.098	0.033	0.065
12	0.098	0.177	0.033	0.066	0.176	0.077
13	0.143	0.253	0.088	0.088	0.010	0.109
14	0.055	0.110	0.066	0.033	0.054	0.055
15	0.088	0.088	0.265	0.297	0.209	0.242
16	0.043	0.110	0.022	0.077	0.220	0.176
17	0.055	0.087	0.010	0.098	0.142	0.066
18	0.121	0.066	0.000	0.121	0.098	0.077
19	0.032	0.132	0.021	0.098	0.143	0.010
20	0.419	0.385	0.253	1.400	1.103	10.250
21	0.044	0.166	0.033	0.166	0.738	1.974
22	0.022	0.055	0.011	0.011	0.033	0.210
23	0.011	0.165	0.055	0.099	0.364	2.835
24	0.066	0.033	0.033	0.055	0.033	1.202
25	0.000	0.043	0.066	0.022	0.120	1.235
26	0.077	0.143	0.076	0.111	1.986	2.652
27	0.088	0.010	0.000	0.043	0.154	0.132
28	0.109	0.176	0.010	0.011	0.772	0.782
29	0.043	0.164	0.055	0.032	0.022	0.032
30	0.011	0.242	0.098	0.043	0.992	0.562
31	0.099	0.253	0.176	2.901	-	-
32	0.210	0.298	0.021	6.719	-	-

33	0.010	0.176	0.143	0.275	7.844	1.202
34	0.055	0.143	0.055	0.032	0.309	0.209
35	0.066	0.087	0.076	6.313	1.852	9.310
36	0.132	0.110	0.154	0.265	-	-
37	0.088	0.198	0.066	0.419	-	-
38	0.022	0.111	0.000	0.010	0.893	0.143
39	0.077	0.142	0.010	0.011	1.290	0.110
40	0.111	0.176	0.618	0.109	-	-
41	1.025	1.412	1.356	0.330	5.628	2.679
42	0.044	0.022	0.066	0.165	5.504	2.853
43	0.032	0.165	0.143	0.098	-	-
44	0.596	-	-	-	0.890	-
45	1.147	1.344	0.309	0.417	-	-
46	2.827	2.195	-	-	-	-
47	0.012	0.132	4.785	-	-	-

Table 5.3: Standard deviation for each algorithm

From the results in table 5.2 it can be concluded that in general ChemPlanner's algorithm with additive heuristic (corresponding to **Additive** column in Table 5.2) performs the best among the other competing algorithms on the benchmark problems. It can also be seen that ChemPlanner's search algorithm guided with additive and max heuristics (corresponding to **Additive** or **Max** columns in Table 5.2) overall perform better than the other search algorithms, specially when the plan length increases. Moreover, the planner that performs problem abstraction and then uses iterative deepening search algorithm to find the abstract plans (corresponding to **ChemPlanner ID** column in Table 5.2) performs better than the planner that works on only concrete level and uses iterative deepening algorithm (corresponding to **ID** column in Table 5.2). Lastly, the planners that work only on the concrete level and employ A^*-like search guided with additive or max heuristics (corresponding to **1L Add** or **1L Max** columns in Table 5.2) perform the weakest among the other competing algorithms, with a slight overall better performance for the search guided with the additive heuristic over the search guided with the max heuristic.

In the ChemPlanner's original algorithm, superiority of additive heuristic over max heuristic can be associated to the fact that additive heuristic does not ignore the cost of achieving any of the preconditions of an action when estimating the heuristic value for a state, while max heuristic ignores all preconditions but one. Superiority of ChemPlanner's algorithm guided by additive and max heuristics over the other algorithms is no surprise either, as it is due to a more sophisticated search algorithm. **ChemPlanner ID** performs better than **ID**, since blind iterative deepening search in the concrete level is computationally prohibitive, while problem abstraction mediates the computation by reducing the search space.

However, it is interesting to note why the algorithms working on one level using A^*-like search guided with delete relaxation heuristics perform worse than the other algorithms in general. The explanation is simple: the delete relaxation heuristics are not suitable for concrete domain of organic chemistry. To elaborate, recall that in delete relaxation heuristics, the negative effects of actions are ignored. In organic chemistry, the actions affect the bonds between the atoms. If bonds are added and never deleted, the graph corresponding to molecules in each situation gets denser and denser, going towards becoming a clique. As a result, many spurious functional groups and chemical classes will be created, which in turn enable many spurious actions to be applicable, which would have not been applicable otherwise. Additionally, there are relatively large number of actions in the macro approach, in our experiments 55 actions. As such, when delete list relaxation heuristics are used on concrete domain of organic chemistry, the branching factor to explore at each state increases considerably and unnecessarily. This results in a weaker performance for this algorithm compared to the other algorithms discussed, specially when the plan length exceeds from one. In some cases, when the plan length is 1, A^*-like search with delete list relaxation heuristic works better than the other algorithms (for example problem 1), which can be explained by noting that this algorithm does not compute high-level molecules in the starting and goal molecules, and therefore saves time.

Experiments with State-of-the-art Planners

The same set of benchmark problems was also translated to PDDL and used in state-of-the-art AI planners in order to determine how well state-of-the-art planners respond to solving organic synthesis problems. Two state of the art planners were experimented. One of them, Fast-Downward [44], is a successful classical planner that employs causal graph heuristics and decompositions of planning tasks augmented with forward-search in order to find the plans. The second, LPG-td planner (Local search for Planning Graphs [36]), is a satisficing planner inspired by WALKSAT, an efficient procedure for solving SAT-problems.

Interestingly, neither of the state-of-the-art planners were capable of solving any of the problems in the benchmark, when at least half of the 55 reactions were included. Both planners exhaust memory in their process, even when experimented on a machine with 128GB of RAM. Fast-Downward, as one of its three stages for planning, translates the PDDL representation of a STRIPS planning task to SAS^+ formalism. It is during this step of planning when Fast-Downward exhausts memory, and therefore falls short of completing finding the plans. Similarly, LPG-td's attempt to solving the problems terminates before finding a solution due to memory exhaustion. To conclude, it appears that the common methodology used in state-of-the-art planners based on building a grounded transition system is not effective on the aforementioned formulation of an organic synthesis problem as a planning task. There are both automatic and manual techniques that can be applied on the problem formulations that can improve the memory requirements of the state-of-the-art planners, such as [4]. In any case, the experiments with the current formulation shed light on some of the shortcomings of the methodologies used in state-of-the art planners, and as such point out some areas that can be improved within the common methodologies adopted by state-of-the-art planners. Needless to say, the performance of ChemPlanner was immensely better than either of the state-of-the-art planners experimented in this work.

The credit for experimenting with state-of-the-art planners goes to Megan Antoniazzi, who worked as a summer student research assistant at the Computer Science department at Ryerson University in the summer of 2014. Also, Vitaliy Batusov wrote a program translating reactions from a chemical representation RXN into PDDL.

5.3.2 Results for the Second Set of Benchmark Problems

Since the knowledge base of chemical reactions used in ChemPlanner subsumes the reactions used in the work of Heifets and Jurisica [41], the benchmark problems used in [41] could also be attempted using ChemPlanner. This section presents and discusses the results obtained from trying the aforementioned benchmark against ChemPlanner.

Table 5.4 displays the times in seconds it took ChemPlanner to identify a plan for each problem, when the problem numbers are kept consistent with those in [41]. The machine conducting the experiments used a 2.30 GHz Intel(R)Core(TM)i7-3610QM CPU, and an allocated 1GB of RAM for the process. Also, a cut off time of 2000 seconds was also in order during the experiments. The problems that are not accounted for in table 5.4 could not be completed in 2000 seconds using either of additive or max heuristics.

Prob. #	Plan length	Additive	Max
2	4	12.901	33.712
5	3	87.064	124.645
10	3	9.656	579.7
19	5	203.097	213.097

Table 5.4: Experimental results on benchmark of Heifets

It is worth noting that results in table 5.4 are not comparable with those reported in [41]. Aside from significant computational power difference between the machines used (Heifets conducts his experiments on an IBM CL1350 cluster with 1,344 cores over 168 Infiniband-connected HS21-XM BladeServers and a DCS9550 storage system, while allowing 8GB of RAM), the reactions used in two works are not exactly comparable either. The reactions in ChemPlanner represent complete effects of each reaction in work of Heifets, and also reasons explicitly about hydrogen atoms, while hydrogen atoms are generally ignored in [41]. As such, ChemPlanner's knowledge base is significantly richer, and as such the planning problems are more complicated.

Yet, it is interesting to ignore the differences in computing powers of the machines used for experimenting, as well as the differences in each problem between our work and that of Heifets, and see how the results compare. Problem number 2 was solved in 7 seconds in [41]. The performance of ChemPlanner, using either additive or max heuristics is not much worse than that, specially when one takes into account the inferiority of the machine that experimented with ChemPlanner. The story is the same for problem number 5, which has been solved in 31 seconds in [41]. Interestingly, ChemPlanner using additive heuristic solves problem 10 much better than how it has been solved in the work of Heifets, when it took 105 seconds to find the solution. Even ChemPlanner with max heuristic is not far behind from what the algorithm in [41] achieved. But most impressive is problem 19, where after 6 hours a solution was not found for it in the work of Heifets, but the planning problem of length 5 was solved in less than four minutes using ChemPlanner, regardless of using additive or max heuristics. Overall, although most of the problems in the benchmark could not be solved within 2000 seconds, the performance of ChemPlanner is not disappointing compared to [41], and in fact promises better performance, at least on some of the problems in the benchmark.

Chapter 6

Conclusion

This chapter concludes this thesis by providing the summary of contributions, some remarks about limitations and future research directions related to this work.

6.1 Summary and Limitations

In this thesis we cast organic synthesis problems as AI planning problems and employed a combination of methods and techniques from the planning community to solve the problems. In more details, we developed a computer-based representation for organic chemistry amenable to reasoning by axiomatizing organic chemistry in SC. That is, we introduced our methodology for expressing arbitrary molecules and functional groups as axioms in SC and elaborated on logical expressivity of the axioms. We also presented, analyzed and compared two approaches to axiomatizing chemical reactions, namely the micro and macro approaches. Both approaches employ the well-defined axioms complying to the requirements in BATs, where successor state axioms are used to represent the effects of the actions and precondition axioms are employed to characterize when actions are applicable.

We then presented a novel algorithm capable of solving organic synthesis problems built by borrowing various techniques from developments in planning community in AI. We used the algorithm to implement an organic synthesis planner, namely ChemPlanner, that has a knowledge base of 55 generic chemical reactions and is capable of identifying 75 functional groups and chemical classes. Effectiveness of ChemPlanner was empirically experimented on two sets of benchmark problems. One set of benchmark problems included 47 synthesis problems of different complexities designed by us. To prove effectiveness of ChemPlanner, the results of ChemPlanner on this benchmark was compared with a number of other competing algorithms, as well as some of the state-of-the-art planners. ChemPlanner was also used to solve problems in a different set of benchmark problems from the literature, and the results were compared with the results reported in the literature.

The solutions generated for an organic synthesis problem in this work are correct in the sense that in theory they are viable. In other words, the reactions in the solution sequence are applicable based on what has been specified as the preconditions, and their cumulative effects result in the goal molecule. However, whether it is guaranteed to work in practice is something that can be verified in a chemistry lab. Additionally, the experimental results in this work demonstrate how long it takes a certain algorithm to solve an organic synthesis problem. Although the timings reported in this work might not be vital to a chemist, meaning a few seconds or minutes longer or shorter wait time to obtain the solution might not be important for a chemist, the results are important from computer science perspective. They demonstrate feasibility of the algorithms and can be used for comparing the algorithms.

The algorithm developed for solving organic synthesis problems, although proved effective in experiments, has some limitations. In short, it converts a concrete synthesis problem to an abstract problem, relies on techniques from planning community to efficiently find solutions for the abstract problem, and then refines the abstract solutions to find a concrete solution for the original problem. It includes a preprocessing phase, performed offline, which computes the high-level effects of reactions in the knowledge base. This phase reaches out to an ontology of chemical classes and molecules. Any limitation and shortcoming in the ontology will be reflected in the high-level effects of the reactions, and possibly can jeopardize the completeness of the planning algorithm.

Moreover, the planning algorithm is most effective when the initial state includes molecules whose chemical classes and functional groups differ from those in the goal molecule. However, if there are extra molecules represented in the initial state, specially those with the same chemical class or functional groups that are present in the goal molecule, the effectiveness of the algorithm is greatly jeopardized. In the latter case, many spurious abstract plans can be identified which cannot be refined to a concrete plan, wasting potentially considerable time on refinement phase. As such, the algorithm is most effective when controlled initial state molecules exist, avoiding extra interfering molecules.

The main limitation of our SC-based representation of chemical molecules and reactions is the lack of ability to represent stereochemistry. Of course, this is not an intrinsic limitation, and proper methods of encoding the knowledge about stereochemistry can be developed and augmented to the current representation. Similarly, the representation can be expanded to include quantitative aspects of chemical reactions (such as yield), which will address the current limitation that only some qualitative aspect of reactions are considered.

6.2 Future Work

Casting organic synthesis as planning in AI opens a large window of opportunities for future work. In this work, we applied some popular techniques from the field of planning to solve synthesis problems. However, the portion of advancements from planning community used in this work is small, and many other ideas and developments can be tried and experimented with. Moreover, it is foreseeable that other novel algorithms for solving synthesis problems can be developed by combining some of the techniques from planning, similar to what was done in this research.

Additionally, despite the encouraging results that were obtained from our experiments, the algorithm of ChemPlanner has to be improved to enhance performance. Further research can help address the limitations of the algorithm described earlier. The current implementation of ChemPlanner can also be improved, as there is room for optimizing the code. The main objective when developing the code for ChemPlanner was its functionality, and developing an optimized code was a second priority. A revisit to the code of ChemPlanner for optimizing it can potentially boost up the performance in orders of magnitude. Other than common sense code optimizations, techniques for optimizing Datalog rules can also be applied to further optimize the program. Such techniques typically re-formulate the rules such that their semantics are kept intact. It is possible that these re-formulations contribute to better performance of state-of-the-art planners as well which did not perform well with the current formulation.

Lastly, systematic development of tools interfacing ChemPlanner with the resources available in the industry remains a future work. There are several databases of chemical reactions and molecules, some of which are publicly available. In order to expand the knowledge base of ChemPlanner, tools translating these databases to the representation used in the ChemPlanner needs to be developed. Preliminary attempt to this end was initiated, but the task was withdrawn as deficiencies in the knowledge bases were identified. Examples of such deficiencies are unbalanced reactions, specific reactions (as opposed to generic), and incomplete representation of the effects of the reactions in most resources available. Another future work, which perhaps requires chemist expert collaboration is to address the deficiencies in the available resources so that they can reliably be included to expand the knowledge base of ChemPlanner.

References

[1] Daylight theory manual, daylight chemical information systems inc. 2011.

[2] `http://www.cs.toronto.edu/~aheifets/ChemicalPlanning/`, accessed on August 10 2014.

[3] Serge Abiteboul, Richard Hull, and Victor Vianu. *Foundations of Databases: The Logical Level*. Addison Wesley, facsimile edition, December 1994.

[4] Carlos Areces, Facundo Bustos, Martín Dominguez, and Jörg Hoffmann. Optimizing planning domains by automatic action schema splitting. In Steve Chien, Minh Binh Do, Alan Fern, and Wheeler Ruml, editors, *ICAPS*. AAAI, 2014.

[5] Sheila Ash, Malcolm A. Cline, R. Webster Homer, Tad Hurst, and Gregory B. Smith. SYBYL line notation (SLN): A versatile language for chemical structure representation. *Journal of Chemical Information and Computer Sciences*, 37(1):71–79, 1997.

[6] Fahiem Bacchus and Qiang Yang. The downward refinement property. In John Mylopoulos and Raymond Reiter, editors, *IJCAI*, pages 286–293. Morgan Kaufmann, 1991.

[7] Fahiem Bacchus and Qiang Yang. Downward refinement and the efficiency of hierarchical problem solving. *Artif. Intell.*, 71(1):43–100, 1994.

[8] Christer Bäckström. Equivalence and tractability results for SAS+ planning. In Bernhard Nebel, Charles Rich, and William R. Swartout, editors, *KR*, pages 126–137. Morgan Kaufmann, 1992.

[9] Christer Bäckström and Peter Jonsson. Bridging the gap between refinement and heuristics in abstraction. In Francesca Rossi, editor, *IJCAI*. IJCAI/AAAI, 2013.

[10] Avrim L. Blum and Merrick L. Furst. Fast planning through planning graph analysis. *Artificial Intelligence*, 90(1):1636–1642, 1995.

[11] Blai Bonet. Abstraction heuristics extended with counting abstractions. In Fahiem Bacchus, Carmel Domshlak, Stefan Edelkamp, and Malte Helmert, editors, *ICAPS*. AAAI, 2011.

[12] Blai Bonet and Hctor Geffner. Planning as heuristic search. *Artificial Intelligence*, 129(12):5 – 33, 2001.

[13] Ronald J. Brachman and Hector J. Levesque. *Knowledge Representation and Reasoning*. Elsevier, 2004.

[14] Tom Bylander. The computational complexity of propositional STRIPS planning. *Artif. Intell.*, 69(1-2):165–204, 1994.

[15] ChemAxon. JChem software. `http://www.chemaxon.com`, accessed on July 20 2014.

[16] William Lingran Chen. Chemoinformatics: Past, present, and future. *Journal of Chemical Information and Modeling*, 46(6):2230–2255, 2006. PMID: 17125167.

[17] Leonid L. Chepelev and Michel Dumontier. Chemical entity semantic specification: Knowledge representation for efficient semantic cheminformatics and facile data integration. *Journal of Cheminformatics*, 3(1):1–19, 2011.

[18] Leonid L. Chepelev, Janna Hastings, Marcus Ennis, Christoph Steinbeck, and Michel Dumontier. Self-organizing ontology of biochemically relevant small molecules. *BMC Bioinformatics*, 13(1):1–18, 2012.

[19] Jens Claen, Gabriele Röger, Gerhard Lakemeyer, and Bernhard Nebel. PLATAS-integrating planning and the action language Golog. *KI - Künstliche Intelligenz*, 26(1):61–67, 2012.

[20] Anthony Cook, A. Peter Johnson, James Law, Mahdi Mirzazadeh, Orr Ravitz, and Aniko Simon. Computer-aided synthesis design: 40 years on. *Wiley Interdisciplinary Reviews: Computational Molecular Science*, 2(1):79–107, 2012.

[21] E. J. Corey and W. Todd Wipke. Computer-assisted design of complex organic syntheses. *American Association for the Advancement of Science*, 166(3902):178–192, 1969.

[22] Joseph C. Culberson and Jonathan Schaeffer. Pattern databases. *Computational Intelligence*, 14(3):318–334, 1998.

[23] Arthur Dalby, James G. Nourse, W. Douglas Hounshell, Ann K. I. Gushurst, David L. Grier, Burton A. Leland, and John Laufer. Description of several chemical structure file formats used by computer programs developed at molecular design limited. *Journal of Chemical Information and Computer Sciences*, 32(3):244–255, 1992.

[24] Evgeny Dantsin, Thomas Eiter, Georg Gottlob, and Andrei Voronkov. Complexity and expressive power of logic programming. *ACM Comput. Surv.*, 33(3):374–425, 2001.

[25] LIACC/Universidade do Porto and at COPPE Sistemas/UFRJ. YAP prolog. `http://www.dcc.fc.up.pt/~vsc/Yap/`, accessed on August 10 2014.

[26] Xue-Min Cheng. Elias James Corey. *The Logic of Chemical Synthesis*. Wiley-Interscience, 1989.

[27] Janna Hastings et al. The ChEBI reference database and ontology for biologically relevant chemistry: enhancements for 2013. *Nucleic Acids Research*, 41(DB):456–463, 2013.

[28] Kirill Degtyarenko et al. ChEBI: a database and ontology for chemical entities of biological interest. *Nucleic Acids Research*, 36(Database-Issue):344–350, 2008.

[29] Richard Fikes and Nils J. Nilsson. STRIPS: A new approach to the application of theorem proving to problem solving. *Artif. Intell.*, 2(3/4):189–208, 1971.

[30] Maria Fox and Derek Long. Pddl2.1: An extension to pddl for expressing temporal planning domains. *Journal of Artificial Intelligence Research*, 20:2003, 2003.

[31] Shinsaku Fujita. Description of organic reactions based on imaginary transition structures. *Journal of Chemical Information and Computer Sciences*, 26(4):205–242, 1986.

[32] Shinsaku Fujita. *Computer-Oriented Representation of Organic Reactions*. Yoshioka Shoten, Kyoto (Japan), 2001.

[33] Johann Gasteiger, Matthias Pförtner, Markus Sitzmann, Robert Höllering, Oliver Sacher, Thomas Kostka, and Norbert Karg. Computer-assisted synthesis and reaction planning in combinatorial chemistry. *Perspectives in Drug Discovery and Design*, 20:245–264, 2000. 10.1023/A:1008745509593.

[34] H. Gelenter, N. S. Sridharan, H. J. Hart, S. C. Yen, F. W. Fowler, and H. J Shue. The discovery of organic synthetic routes by computer. *Topics Curr. Chem.*, 41:113, 1973.

[35] Herbert Gelernter, J. Royce Rose, and Chyouhwa Chen. Building and refining a knowledge base for synthetic organic chemistry via the methodology of inductive and deductive machine learning. *Journal of Chemical Information and Computer Sciences*, 30(4):492–504, 1990.

[36] Alfonso Gerevini, Alessandro Saetti, Ivan Serina, and Paolo Toninelli. LPG-TD: a fully automated planner for PDDL2.2 domains. In *In Proc. of the 14th Int. Conference on Automated Planning and Scheduling (ICAPS-04) International Planning Competition abstracts*, 2004.

[37] Wen-Xiang Gu, Jin-Li Li, Ming-Hao Yin, Jun-Shu Wang, and Jin-Yan Wang. A novel causal graph based heuristic for solving planning problem. In *Machine Learning and Cybernetics, 2008 International Conference on*, volume 4, pages 2223–2228, July 2008.

[38] Norbert Haider. Functionality Pattern Matching as an Efficient Complementary Structure/Reaction Search Tool: an Open-Source Approach. *Molecules*, 15(8):5079–5092, July 2010.

[39] S. Hanessian. Man, machine and visual imagery in strategic synthesis planning: Computer-perceived precursors for drug candidates. *Curr. Opin. Drug Discovery Dev.*, 8:798, 2005.

[40] Janna Hastings, Michel Dumontier, Duncan Hull, Matthew Horridge, Christoph Steinbeck, Ulrike Sattler, Robert Stevens, Tertia Hörne, and Katarina Britz. Representing chemicals using owl, description graphs and rules.

[41] Abraham Heifets and Igor Jurisica. Construction of new medicines via game proof search. In Jörg Hoffmann and Bart Selman, editors, *AAAI*, pages 1564–1570. AAAI Press, 2012.

[42] Stephen Heller, Alan McNaught, Stephen Stein, Dmitrii Tchekhovskoi, and Igor Pletnev. InChI - the world-wide chemical structure identifier standard. *Journal of Cheminformatics*, 5(1), 2013.

[43] Malte Helmert. A planning heuristic based on causal graph analysis. In Shlomo Zilberstein, Jana Koehler, and Sven Koenig, editors, *ICAPS*, pages 161–170. AAAI, 2004.

[44] Malte Helmert. The fast downward planning system. *Journal of Artificial Intelligence Research*, 26:191–246, 2006.

[45] Malte Helmert. Concise finite-domain representations for PDDL planning tasks. *Artificial Intelligence*, 173(56):503 – 535, 2009.

[46] Malte Helmert and Carmel Domshlak. Landmarks, critical paths and abstractions: What's the difference anyway? In Alfonso Gerevini, Adele E. Howe, Amedeo Cesta, and Ioannis Refanidis, editors, *ICAPS*. AAAI, 2009.

[47] Malte Helmert and Hector Geffner. Unifying the causal graph and additive heuristics. In Jussi Rintanen, Bernhard Nebel, J. Christopher Beck, and Eric A. Hansen, editors, *ICAPS*, pages 140–147. AAAI, 2008.

[48] Malte Helmert, Patrik Haslum, and Jörg Hoffmann. Flexible abstraction heuristics for optimal sequential planning. In Mark S. Boddy, Maria Fox, and Sylvie Thiébaux, editors, *ICAPS*, pages 176–183. AAAI, 2007.

[49] Jörg Hoffmann. FF: The fast-forward planning system. *AI magazine*, 22:57–62, 2001.

[50] Jörg Hoffmann. Analyzing search topology without running any search: on the connection between causal graphs and h+. *J. Artif. Int. Res.*, 41(2):155–229, May 2011.

[51] Jörg Hoffmann and Bernhard Nebel. The FF planning system: Fast plan generation through heuristic search. *Journal of Artificial Intelligence Research*, 14:2001, 2001.

[52] Qi Huang, Lin-Li Li, and Sheng-Yong Yang. RASA: A rapid retrosynthesis-based scoring method for the assessment of synthetic accessibility of drug-like molecules. *Journal of Chemical Information and Modeling*, 51(10):2768–2777, 2011.

[53] Michael Katz, Jörg Hoffmann, and Carmel Domshlak. Red-black relaxed plan heuristics. In Marie desJardins and Michael L. Littman, editors, *AAAI*. AAAI Press, 2013.

[54] Craig A. Knoblock. Automatically generating abstractions for planning. *Artif. Intell.*, 68(2):243–302, August 1994.

[55] Mykola Konyk, Alexander Leon, and Michel Dumontier. Chemical knowledge for the semantic web. In Amos Bairoch, Sarah Cohen-Boulakia, and Christine Froidevaux, editors, *Data Integration in the Life Sciences*, volume 5109 of *Lecture Notes in Computer Science*, pages 169–176. Springer Berlin Heidelberg, 2008.

[56] Masaaki Kotera, Andrew G. McDonald, Sinad Boyce, and Keith F. Tipton. Functional group and substructure searching as a tool in metabolomics. *PLoS ONE*, 3(2):e1537, 02 2008.

[57] Oliver Kutz, Janna Hastings, and Till Mossakowski. Modelling highly symmetrical molecules: Linking ontologies and graphs. In Allan Ramsay and Gennady Agre, editors, *Artificial Intelligence: Methodology, Systems, and Applications*, volume 7557 of *Lecture Notes in Computer Science*, pages 103–111. Springer Berlin Heidelberg, 2012.

[58] Bradford John Larsen, Ethan Burns, Wheeler Ruml, and Robert Holte. Searching without a heuristic: Efficient use of abstraction. In Maria Fox and David Poole, editors, *AAAI*. AAAI Press, 2010.

[59] James Law, Zsolt Zsoldos, Aniko Simon, Darryl Reid, Yang Liu, Sing Yoong Khew, A. Peter Johnson, Sarah Major, Robert A. Wade, and Howard Y. Ando. Route designer: A retrosynthetic analysis tool utilizing automated retrosynthetic rule generation. *Journal of Chemical Information and Modeling*, 49(3):593–602, 2009.

[60] Hector J. Levesque, Fiora Pirri, and Raymond Reiter. Foundations for the situation calculus. *Electron. Trans. Artif. Intell.*, 2:159–178, 1998.

[61] Vladimir Lifschitz. On the semantics of STRIPS. In Michael Georgeff, Lansky, and Amy, editors, *Reasoning about Actions and Plans*, pages 1–9. Morgan Kaufmann, San Mateo, CA, 1987.

[62] Fangzen Lin and Raymond Reiter. State constraints revisited. *Journal of Logic and Computation*, 4(5):655–677, 1994.

[63] Fangzhen Lin and Raymond Reiter. How to progress a database. *Artificial Intelligence*, 92(12):131 – 167, 1997.

[64] Yongmei Liu and Gerhard Lakemeyer. On first-order definability and computability of progression for local-effect actions and beyond. In *Proceedings of the 21st international joint conference on Artificial intelligence*, IJCAI'09, pages 860–866, San Francisco, CA, USA, 2009. Morgan Kaufmann Publishers Inc.

[65] Yongmei Liu and Hector J. Levesque. A tractability result for reasoning with incomplete first-order knowledge bases. In *proceedings of IJCAI-03*, pages 83–88, 1998.

[66] John W. Lloyd. *Foundations of Logic Programming, 2nd Edition*. Springer, 1987.

[67] Despoina Magka, Boris Motik, and Ian Horrocks. Modelling structured domains using description graphs and logic programming. In Elena Simperl, Philipp Cimiano, Axel Polleres, Oscar Corcho, and Valentina Presutti, editors, *The Semantic Web: Research and Applications*, volume 7295 of *Lecture Notes in Computer Science*, pages 330–344. Springer Berlin Heidelberg, 2012.

[68] Arman Masoumi, Andrea Marrella, and Mikhail Soutchanski. Towards a planning-based approach to the automated design of chemical processes. In Laura Giordano, Stefania Montani, and Daniele Theseider Dupré, editors, *AIBP@AI*IA*, volume 1101 of *CEUR Workshop Proceedings*, pages 61–70. CEUR-WS.org, 2013.

[69] Arman Masoumi and Mikhail Soutchanski. Reasoning about chemical reactions using the situation calculus. In *AAAI Fall Symposium: Discovery Informatics*, volume FS-12-03 of *AAAI Technical Report*. AAAI, 2012.

[70] Arman Masoumi, Mikhail Soutchanski, and Andrea Marrella. Organic synthesis as artificial intelligence planning. In Adrian Paschke, Albert Burger, Paolo Romano, M. Scott Marshall, and Andrea Splendiani, editors, *SWAT4LS*, volume 1114 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2013.

[71] John McCarthy. Formalization of STRIPS in situation calculus. *Technical Report Formal Reasoning Group, Department of Computer Science, Stanford University*, 1985.

[72] Drew McDermott. Using regression-match graphs to control search in planning. *Artificial Intelligence*, 109(12):111 – 159, 1999.

[73] Drew Mcdermott. The 1998 AI planning systems competition. *AI Magazine*, 21:35–55, 2000.

[74] Sheila A. McIlraith. Integrating actions and state constraints: A closed-form solution to the ramification problem (sometimes). *Artificial Intelligence*, 116(12):87 – 121, 2000.

[75] Allen Newell, J. C. Shaw, and Herbert A. Simon. Report on a general problem-solving program. In *IFIP Congress*, pages 256–264, 1959.

[76] M. Ott. Cheminformatics and organic chemistry. computer-assisted synthetic analysis. *Cheminformatics*, 1:83, 2004.

[77] Frederic Pennerath, Gilles Niel, Philippe Vismara, Philippe Jauffret, Claude Laurenco, and Amedeo Napoli. Graph-mining algorithm for the evaluation of bond formability. *Journal of Chemical Information and Modeling*, 50(2):221–239, 2010. PMID: 20112969.

[78] G. Polya. *How to Solve It.* Princeton University Press, November 1971.

[79] Ewgenij Proschak, Jörg K. Wegner, Andreas Schüller, Gisbert Schneider, and Uli Fechner. Molecular query language (MQL) a context-free grammar for substructure matching. *Journal of Chemical Information and Modeling*, 47(2):295–301, 2007.

[80] Raymond Reiter. *Knowledge in Action. Logical Foundations for Specifying and Implementing Dynamical Systems.* MIT, 2001.

[81] Francesc Rosselló and Gabriel Valiente. Analysis of metabolic pathways by graph transformation, in. In *Proc. 2 nd Int. Conf. Graph Transformation, Lect*, pages 70–82, 2004.

[82] Francesc Rosselló and Gabriel Valiente. Chemical graphs, chemical reaction graphs, and chemical graph transformation. *Electron. Notes Theor. Comput. Sci.*, 127(1):157–166, March 2005.

[83] Earl D. Sacerdoti. Planning in a hierarchy of abstraction spaces. *Artificial Intelligence*, 5(2):115 – 135, 1974.

[84] Punnaivanam Sankar, Alain Krief, and Durairaj Vijayasarathi. A conceptual basis to encode and detect organic functional groups in XML. *Journal of Molecular Graphics and Modelling*, 43(0):1 – 10, 2013.

[85] Josefina Sierra-Santibez. A declarative formalization of STRIPS. In *In Proceedings of the 13th European Conference on Artificial Intelligence, ECAI-98, John Wiley and Sons Ltd*, pages 509–513, 1998.

[86] Terrance Swift and David S. Warren. XSB: Extending prolog with tabled logic programming. *Theory Pract. Log. Program.*, 12(1-2):157–187, January 2012.

[87] Carolyn L. Talcott. Pathway logic. In Marco Bernardo, Pierpaolo Degano, and Gianluigi Zavattaro, editors, *SFM*, volume 5016 of *Lecture Notes in Computer Science*, pages 21–53. Springer, 2008.

[88] Akio Tanaka, Hideho Okamoto, and Malcolm Bersohn. Construction of functional group reactivity database under various reaction conditions automatically extracted from reaction database in a synthesis design system. *Journal of Chemical Information and Modeling*, 50(3):327–338, 2010. PMID: 20187659.

[89] Josh Tenenberg. *Abstraction in Planning*. PhD thesis, University of Rochester, Department of Computer Science, 1988.

[90] Sylvie Thibaux, Jörg Hoffmann, and Bernhard Nebel. In defense of PDDL axioms. *Artificial Intelligence*, 168(12):38 – 69, 2005.

[91] Matthew H. Todd. Computer-aided organic synthesis. *Chem. Soc. Rev.*, 34:247–266, 2005.

[92] Serge S. Tratch and Nikolai S. Zefirov. Systematic search for new types of chemical interconversions: Mathematical models and some applications. *Journal of Chemical Information and Computer Sciences*, 38(3):331–348, 1998.

[93] Raúl E. Valdés-Pérez. Machine Discovery in Chemistry: New Results. *Artif. Intell.*, 74(1):191–201, 1995.

[94] G. E. Vléduts. Concerning one system of classification and codification of organic reactions. *Inf. Storage Retr.*, 1:117, 1963.

[95] G. E. Vléduts. Do we still need a classification of reactions? In Peter Willett, editor, *Modern Approaches to Chemical Reaction Searching, Proceedings of a Conference by the Chemical Structure Association of the University of York, England, 8-11 July 1985*, pages 202–220, Aldershot, England, 1986. Gower Publishing Company.

[96] G. E. Vléduts and E. A. Geivandov. *Automated Information Systems for Chemistry (in Russain)*. Nauka, Moscow, 1974.

[97] William J. Wiswesser. How the WLN began in 1949 and how it might be in 1999. *Journal of Chemical Information and Computer Sciences*, 22(2):88–93, 1982.

[98] Nikolai S. Zefirov, Igor I. Baskin, and Vladimir A. Palyulin. SYMBEQ program and its application in computer-assisted reaction design. *Journal of Chemical Information and Computer Sciences*, 34(4):994–999, 1994.

[99] Steven S. Zumdahl. *Introductory Chemistry: A Foundation*. Houghton Mifflin Col, 2003.