Exploring Organic Synthesis with State-of-the-Art Planning Techniques

Rami Matloob and **Mikhail Soutchanski**
Dept. of Comp. Science,
Ryerson University, 245 Church Street, ENG281, Toronto, ON, M5B 2K3, Canada

Abstract

We explore different techniques to solve the computationally challenging, but practically important organic synthesis problem. This problem requires finding a sequence of reactions producing the target molecule from a set of given initial molecules. This problem is often used on exams to test the problem solving skills of students who study generic reactions in organic chemistry courses. This problem is also relevant in industry. In a quest to find a more efficient way to solve a set of benchmark problems, we start by explaining how the organic synthesis problem can be formulated as a planning problem in PDDL. We then demonstrate how state-of-the-art planners such as SASE, Madagascar and SatPlan – that reduce a bounded planning problem to satisfiability – can only encode a fraction of the benchmark problems. We also explore the recently developed action schema splitting syntactic transformation that splits each action into several sub-actions with a shorter interface thereby alleviating somewhat the grounding problem. We assess experimentally the performance of the Fast Downward planning system for different splitting values and compare the results with our original un-split domain. For the unique finest split, where the modified action schemas have the smallest number of arguments, we investigate performance of Fast Downward with different heuristics. We propose organic synthesis as the new challenge for planning.

1 Introduction

We are involved into a long-term project related to developing an online automated tutoring system for undergraduate students who study Organic Chemistry. The goal is to create a system that will provide many functionalities. In particular, the main feature will be helping the chemistry students with solving the multi-step organic synthesis problem. The organic synthesis problem is a common chemistry problem that requires finding a sequence of reactions that produces a target molecule from given initial molecules. The students who take Organic Chemistry courses are introduced to many generic reactions. These reactions are generic in the sense that they are applicable to multiple classes of molecules. The students study details of each reaction including how atoms in participating molecules change bonds. To test the students knowledge it is common to give them a few organic synthesis problems so the students can discover the right combination of reactions for a given target molecule and initial molecules. These exam problems can vary in difficulty from relatively simple problems that can be solved in 2 or 3 steps to more complex problems that may require finding a sequence of 10-12 reactions. The students can improve their knowledge about reactions if they can practice solving numerous organic synthesis problems before the exam. We would like to develop a tool that will be verifying whether a solution proposed by a student is correct, and if not, after a number of unsuccessful attempts, show a correct solution upon request from the student. More specifically, we would like to develop a tool that can be challenged with any organic synthesis problem. The problem can be either entered by a human using existing open source molecule editing tools, or it can be generated automatically. The range of organic synthesis problems is potentially unlimited, i.e., the tool should be able to solve any problem of certain level of complexity, but not just problems out of a finite library entered in advance. It is possible that this tool will be also helpful for research purposes since the organic synthesis problem is a long-standing problem with industrial applications. There has been a long history of research related to *computer aided synthesis design* (CASD). This history and the current developments are well outlined in (Judson 2009; Cook et al. 2012; Ravitz 2013; Bøgevig et al. 2015; Szymkuć et al. 2016). Reviewing this history would not be relevant for the purposes of our paper, but it is sufficient to mention that most of the systems required interaction with a human assistant, and to the best of our knowledge they are either no longer developed, or they are not publicly available for assessment.

The recent work (Heifets and Jurisica 2012) explored solving the organic synthesis problem using AND/OR graph search with a modified proof number search (this approach is popular when solving large two-player zero-sum games such as Go or Chess). The significant contributions of their work include development of 20 benchmark problems inspired by the exam questions given to the students at the Massachusetts Institute of Technology (MIT) taking Course 5.13 Organic Chemistry II during 2001-2006. Since these are real exam questions, it is interesting to see if they can be solved by a computer program. About 50 reactions required to solve all these 20 problems were manually encoded using a specialized chem-informatics language (James, Weininger, and Delany 2011). This benchmark set is publicly available from (Heifets 2012). The search for a sequence of reactions was facilitated by a proprietary chem-informatics software JChem developed by *ChemAxon*; it reads encodings of reactions and molecules and can answer queries whether molecules match (ChemAxon 2015). Since this search requires significant computational resources, it was implemented on an IBM super-computer. Each test received up to 6 hours of CPU time and 8GB of RAM to solve one of the benchmark problems. The experimental results are reported in (Heifets and Jurisica 2012). They demonstrate that some of the 20 benchmark problems could be solved within 6 hours limit, but the problems 16, 17, 18, 19 and 20 could not be solved.

This previous research begs naturally the question whether the organic synthesis problems can be solved using any of the modern AI planning techniques while relying only on standard hardware. To explore this, (Masoumi, Antoniazzi, and Soutchanski 2015) has developed representation of reactions in the Planning Domain Definition Language (PDDL). PDDL has been designed to standardize Artificial Intelligence (AI) planning languages. It is extensively used by all systems competing at the International Planning Competitions. More specifically, (Masoumi, Antoniazzi, and Soutchanski 2015) has developed PDDL representation of generic reactions in PDDL 2.2 that includes derived predicates (Edelkamp and Hoffmann 2004). Before any of the planning problems could be solved, both initial and goal states should be encoded in PDDL. This has been done manually. Additionally, the reactions from the benchmark (Heifets 2012) should be somehow translated from the specialized chem-informatics language into PDDL. Automatic translation was impossible because some effects of the reactions in (Heifets 2012) are left unspecified. This was not an obstacle for research reported in (Heifets and Jurisica 2012) since their program searched from target to input molecules. However, in PDDL encoding, all the effects must be stated explicitly. For this reason, all the required reactions have been manually entered and saved using *MarvinSketch* reaction editing software provided by *ChemAxon* (ChemAxon 2015). This software allows saving the reactions in a RXN format that is popular in chem-informatics industry (Accelrys 2011). Subsequently, the reactions have been automatically translated from RXN into PDDL using an in-house developed computer program. Experimental results reported in (Masoumi, Antoniazzi, and Soutchanski 2015) are mostly negative because the evaluated AI planners suffered from the grounding problem. Each planning instance involves a few dozen atoms, such as carbon, hydrogen or oxygen, while some of the actions have more than 5 arguments, and as a consequence, when actions are instantiated with available atoms the size of a transition system constructed in the process of grounding exceeds the available memory. As reported in (Masoumi, Antoniazzi, and Soutchanski 2015), the computer memory with 128GB of RAM was not enough even if only 23 actions remained in the PDDL domain.

Several promising research directions have been identified in (Masoumi, Antoniazzi, and Soutchanski 2015). Our paper reports experimental results collected while pursuing some of the identified research directions. This is the first contribution of our paper. A close inspection of the previously developed PDDL domain with reactions has revealed that some of the reactions and/or planning instances with initial and goal states had inaccuracies, e.g., types mismatches. Some of the errors occurred because the reactions have been entered into *MarvinSketch* manually by a Computer Science undergraduate student. For this reason, before we could proceed, we have carefully debugged a set of reactions. The resulting clean set of 24 reactions is the PDDL domain that we use in our research: see Table 1. This domain can be used to solve some of the benchmark problems: see Table 2. For each of these problems, we have verified that it can actually be solved using the reactions from a smaller sub-domain that includes only required reactions. Thanks to this verification, we are confident that all problems we are investigating in this paper actually have a solution that can be potentially computed. Elucidating this clean PDDL sub-domain is our second contribution.[1]

Reaction Name	#VARs	PRE	ADD	DEL
alcoholAndPBr3	7	8	4	4
aldolCondensation	15	22	10	10
alkeneAndWater	5	5	6	4
alkylHalideAndCyanide	5	4	4	4
anhydrideReduction	17	46	16	20
aromaticNitration	15	25	6	6
carboxylicAcidAndThionylChloride	9	10	6	8
catalyticHydrogenationOfNitroGroup	6	6	6	6
dediazoniation	6	6	2	4
diazotization	10	13	4	10
dielsAlder	6	19	12	8
enolSN2attackOnAlkylHalide	6	8	8	8
grignard	6	4	6	4
grignardAdditionToAcidChloride	7	7	4	4
grignardReagentFormation	3	1	4	2
imineFormation	8	9	6	6
imineReductionToAmine	14	22	8	6
ketoneAndLDA	13	29	8	8
ketoneReduction	10	17	6	6
michaelAdditionWithKetones	9	15	6	4
michaelAddition	7	14	6	4
oxidationOfAlcoholsWithPCC	16	30	8	10
nitrileReductionToAmine	9	14	10	12
sandmeyerReaction	6	5	4	6

Table 1: 24 verified reactions and their parameters.

Pr	TotalObj	Hydrogen	Carbon	Oxygen	PlanLength
2	40	16	8	11	4
4	68	24	9	21	8
5	62	28	20	8	3
6	32	10	4	5	7
10	59	31	14	7	3
14	74	40	17	5	5
17	94	46	27	12	5
20	58	22	9	6	11

Table 2: Number of objects in each problem and plan length

The rest of our paper is structured as follows. First, we review relevant work from planning and provide introduction to organic chemistry to facilitate understanding of our paper. Second, we report results from our experimental evaluation. They include attempts to solve the benchmark problems using programs that reduce planning to satisfiability. Subsequently, we have attempted to transform the domain syntactically using splitting approach developed in (Areces et al. 2014). We report results collected with Fast Downward (FD) software (Helmert and et al 2015) that we use to solve benchmark problems with the domains produced by splitting software. As a conclusion, we discuss some of the research directions that can be further explored. It remains to be seen whether modern AI planning techniques are mature enough to compete with undergraduate students when solving exam problems from Organic Chemistry.

[1]We are going to share publicly the PDDL encoding of all reactions and all 20 benchmark problems once we have completed debugging and verified that there are no errors left.

2 Background

The following papers investigate how reducing a bounded planning problem to satisfiability (SAT) can be an efficient approach to solving planning problems. In 1992, (Kautz and Selman 1992) proposed reduction of planning to satisfiability and explained how effects and preconditions can be encoded into SAT. This paper also explains how restrictions can be added in SAT so encodings can be more compact.

The details and the advantages of linear encodings (with operator splitting and explanatory frame axioms) and parallelized encodings are explained in (Kautz, McAllester, and Selman 1996). This is also the first paper that introduces lifted causal encodings.

The idea of reducing classical planning problems efficiently to SAT is further investigated in (Ernst, Millstein, and Weld 1997). This is the first paper that discusses the features of a fully-implemented compiler that can generate SAT encodings for STRIPS planning problems.

SatPlan's technical guide (Kautz, Selman, and Hoffmann 2006) discusses the different versions of the SatPlan system and how it has improved over the years.

A new encoding scheme based on the SAS+ formalism is introduced in (Huang, Chen, and Zhang 2010). Their encoding reduces the number of clauses in the problem by exploiting the structural information in the SAS+ formalism, where each variable can vary over a finite domain, while PDDL has only boolean variables. This paper also introduces a new SAT-based planner named SASE.

Madagascar is a modern SAT-based planner that was implemented to improve scalability of large SAT problems (Rintanen 2014; 2015).

Action schema splitting automated domain transformation approach is introduced in (Areces et al. 2014). It transforms an action schema with a big interface (many parameters) into several properly coordinated sub-actions with smaller interfaces. This automated transformation helps reducing the number of ground actions, but makes the length of a plan longer. The grain of split (course split vs fine split) can be controlled in an implementation using a numerical parameter γ. The smaller γ is, the fewer variables the generated sub-actions will have, and the more sub-actions will be produced for each given action schema. We use an implementation of this approach downloaded from the author's Web site on September 19, 2014.

3 Preliminaries

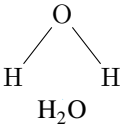
In this Section, we explain how a chemical molecules and reactions can be encoded in PDDL. Consider molecules as graphs with edges that have four different labels representing four common types of bonds between atoms. We model them using the predicate $bond(?x, ?y)$ – it represents a single bond between atoms $?x$ and $?y$ – and the predicates $doublebond(?x, ?y)$, $triplebond(?x, ?y)$ and $aromaticbond(?x, ?y)$, where the latter bond is for the case when several atoms share electrons as in the benzene ring molecule. Furthermore, in graphs representing molecules, vertices are labeled by the name of a chemical atom from the periodic table, such as carbon (C), oxygen (O), hydrogen (H), nitrogen (N), sodium (Na), chromium (Cr), chlorine (Cl), and so on. Since molecules can be described as graphs, it is convenient to consider reactions as graph transformations that break some of the existing bonds and form some new bonds between atoms in participating molecules. Each chemical atom has valence, which can be determined by the number of connections it can form. For example, hydrogen has a valence of 1, oxygen has a valence of 2, and therefore oxygen can form one double bond, or two single bonds. Each carbon atom has a valence of 4, and for this reason, it can form all four kinds of bonds as explained above, e.g., one triple bond and one single bond, or one double bond and two single bonds, and so on. When drawing graphs representing molecules with aromatic bonds, it is common to draw a circle to portray these bonds for atoms around the circle, e.g., see the pyridine molecule C_5H_5N that occurs before and after reaction arrow in Figure 1. For historical reasons, aromatic bonds are sometimes drawn as an alternating sequence of single and double bonds (see Appendix).

To explain our encoding of reactions in PDDL, we use problem 5 as an example. One of molecules participating in reactions solving problem 5 is water or H_2O. We define the atoms in H_2O as follows:

```
o51 - oxygen
h51 - hydrogen
h52 - hydrogen
```

where we write first names of the constants and then their types. To describe bonds between the atoms in the molecule H_2O we have to write each predicate twice to characterize bonds in both directions (for clarity, we do not show numerical indexes in the illustrations).

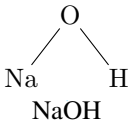
(bond h51 o51) ;
(bond o51 h51) ;
(bond h52 o51) ;
(bond o51 h52) ;



Another participating molecule sodium hydroxide or NaOH can be represented as follows.

```
na - sodium
o50 - oxygen
h50 - hydrogen
```

(bond na o50) ;
(bond o50 na) ;
(bond h50 o50) ;
(bond o50 h50) ;



The problem 5, abbreviated subsequently as p5, can be solved using combination or two reactions; first, $oxidationOfAlcoholsWithPCC$ is executed twice with different molecules, and then $aldolCondensation$ is executed last to produce the target molecule. Figures 1 and 2 with molecules participating in the reactions are shown below. In RXN files all atoms are consecutively numbered. In the figures, the numbers are adjacent to each atom. Thanks to these numbers, it is possible to identify the changes in bonds between atoms during the reactions. The complete PDDL code for $aldolCondensation$ can be found in Appendix, but below we show and discuss a snippet that includes a few preconditions and selected effects, for brevity.

We encode generic reactions as action schemas because reactions apply to large classes of molecules not just to specific molecules. For example, many molecules in organic

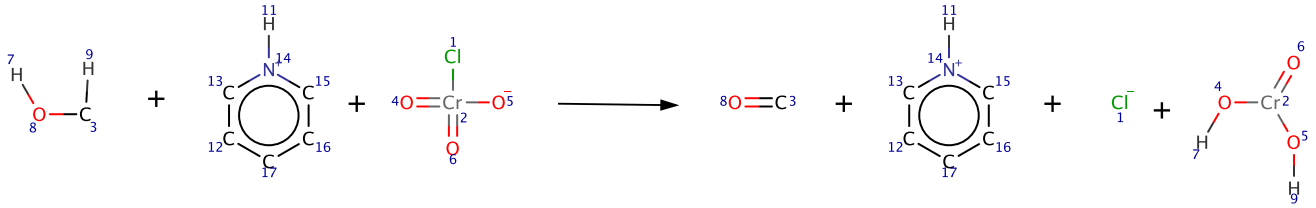


Figure 1: Oxidation of alcohols with PCC reaction

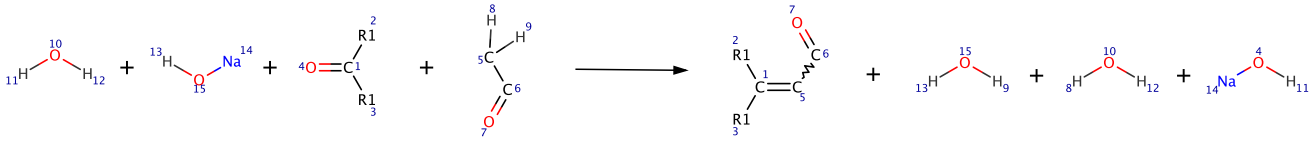


Figure 2: Aldol condensation reaction

chemistry use alkyls. The simplest alkyls are methyl $-CH_3$ and ethyl $-CH_2-CH_3$. Each alkyl has a *key carbon atom* which bonds with an external atom to form a molecule. An alkyl is a molecule with the general formula C_nH_{2n+1}, where n is a positive integer. The alkyls are arbitrary branching trees of single bonded carbon atoms such that the remaining carbon valences are filled with hydrogens. In Figure 2, alkyls are denoted with $R1$, but each $R1$-instance can be a different alkyl. Therefore, $aldolCondensation$ is a schema covering a class of individual reactions. In our program, alkyls can be represented as R-group atoms or can be simply represented by a single key carbon atom in the PDDL code. In (Masoumi, Antoniazzi, and Soutchanski 2015), it was attempted to use PDDL derived predicates to represent alkyls and other functional groups common in organic chemistry, but this did not work since grounding of the PDDL domain with derived predicates could not fit in memory.

In $aldolCondensation$, the most important change involves two bonds. On the left hand side, the double bond between the carbon atom with number 1 and the oxygen atom with the number 4 cleaves, and subsequently, the 4th atom forms new bonds with potassium Na (number 14) and hydrogen (number 11). On the right hand side, the new double bond between the 1st and 5th carbon atoms forms, while the previous bonds of the 5th carbon with 8th and 9th hydrogen atoms cleave. In the PDDL code snippet we keep only the main changes, while skipping minor details. The indexes of PDDL variables correspond to the numbers assigned to atoms in the figure. We can describe each bond once when we write preconditions because bonds are symmetrical in the input molecules, but to specify effects we write each bond relation in both directions to maintain its symmetry.

```
(:action aldolCondensation
    :parameters  /* skip for brevity */
    :precondition (and (not (= ?c_1 ?c_5))
(not (= ?o_10 ?o_4))  (not (= ?o_10 ?o_15))
(not (= ?o_4 ?o_15)) (not (= ?r1_2 ?r1_3))
 (bond ?r1_2 ?c_1) (bond ?r1_3 ?c_1)
(doublebond ?o_4 ?c_1) /* skip a few */
(not (= ?h_9 ?h_8)) (not (= ?c_6 ?c_5))
(bond ?c_6 ?c_5) (bond ?h_9 ?c_5)
(bond ?h_8 ?c_5) (doublebond ?o_7 ?c_6))
    :effect (and /* skip some effects */
(not(doublebond ?o_4 ?c_1))(not(doublebond ?c_1 ?o_4))
 (not (bond ?h_8 ?c_5)) (not (bond ?c_5 ?h_8))
 (not (bond ?h_9 ?c_5)) (not (bond ?c_5 ?h_9))
            )
)
```

In the $oxidationOfAlcoholsWithPCC$ reaction, the large middle pyridinium molecule C_5H_5NH is part of pyridinium chlorochromate (PCC) reagent $C_5H_5NH[CrO_3Cl]$. Its purpose is transforming the left-most alcohol molecule into the carbonyl molecule C=O formed by the the 3rd atom (carbon) and the 8th atom (oxygen). This carbon atom stands for an alkyl that before reaction bonds with hydroxyl -OH thereby forming an alcohol molecule. In the products, we see that the 7th atom (hydrogen from hydroxyl) cleaves from 8th atom (oxygen) and forms a new bond. However, we skip this and other minor changes in the PDDL code snippet below to focus on the purpose of this reaction.

```
(:action oxidationOfAlcoholsWithPCC
    :precondition (and (not (= ?h_9 ?h_7))
(not (= ?o_8 ?o_5)) (not (= ?o_8 ?o_4))
(not (= ?o_5 ?o_4)) (bond ?c_3 ?h_9)
 (bond ?c_3 ?o_8) (bond ?o_8 ?h_7)
  /* skip other preconditions */
                )
    :effect (and /* skip some effects */
(not (bond ?c_3 ?h_9)) (not (bond ?h_9 ?c_3))
(not (bond ?c_3 ?o_8)) (not (bond ?o_8 ?c_3))
(doublebond ?c_3 ?o_8)  (doublebond ?o_8 ?c_3)
(bond ?o_5 ?h_9) (bond ?h_9 ?o_5)
(not (bond ?o_8 ?h_7)) (not (bond ?h_7 ?o_8))
(bond ?o_4 ?h_7) (bond ?h_7 ?o_4)
            )
)
```

As described in Table 1, $oxidationOfAlcoholsWithPCC$

involves 16 parameters, while $aldolCondensation$ has 15 parameters. Since the initial state of the problem 5 has 62 atoms (see Table 2), instantiation of these 2 reactions requires significant memory. Therefore, when we took only 2 reactions required to solve p5, FD was not able to solve this problem on a computer with 32GB of memory, but could solve it when we eliminated from a reaction one of the hydrogen atoms that has been non-essential for this planning instance. Subsequently, we call this engineered version as a "p5 with missing hydrogen" problem.

4 Experiments

4.1 SAT-Based Planners

We use 3 different planners, SASE (Huang, Chen, and Zhang 2010), the version released on August 30, 2011, SatPlan (Kautz, Selman, and Hoffmann 2006), the version released in 2006, and Madagascar, version M (with default parameters), (Rintanen 2014; 2015), publicly available as a C code with the last modifications from February 2015, to encode and solve problems. For brevity, we refer to these 3 programs as "encoders" because they encode a given planning instance as SAT. The produced encodings were compared based on the number of variables, denoted as NoV, and number of clauses, denoted as NoC. However, we did not try to feed encodings to any of the SAT solvers.

Note that SatPlan gives the option to use 4 different encodings and they are: 1)action-based encoding, 2)gpstyle-action-based encoding, 3) gp-based encoding and 4) thin-bp-based encoding.

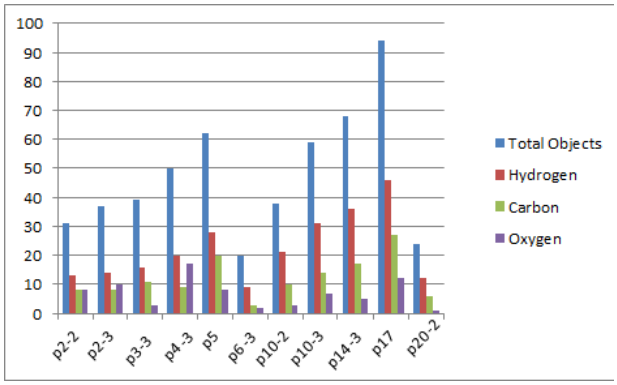


Figure 3: Diagram for number of atoms

Encoding Directly as CNF We started gathering results from Madagascar, SASE and SatPlan by generating a domain file that contained all the actions needed for the problems mentioned in Table 2. All experiments were performed on a server running a virtual machine with 2.80 GHz CPU, Ubuntu 14.04 and 128GB of RAM. The results were not very promising for the full problems. We were able to only get CNF encodings of some of the problems. All three encoders, either produced a very large CNF encoding or ran out of memory (Madagascar) when trying to generate an encoding. Table 3 and Table 4 show more information about the encodings generated by SASE and SatPlan, respectively.

Problem	NoV	NoC
P2-Full	10079	782197
P4-Full	9784	742043
P5-Full-Hydrogen missing	460189	101711688
P6-Full	3148	95580
P10-1Step	132	340
P14-Full	2856	121267
P17-Full	out of memory	out of memory
P20-Full	15331	610997

Table 3: CNF encodings generated by SASE

Problem	NoV	NoC	Encoding
P2-Full	Not solved	Not solved	
P4-Full	Not solved	Not solved	
P5-Missing-H	Not solved	Not solved	
P6-Full	4318	879100	1
P6-Full	3958	1976336	2
P6-Full	4928	2027910	3
P6-Full	4928	713534	4
P10-1Step	12	50	1
P10-1Step	12	50	2
P10-1Step	44	242	3
P10-1Step	44	242	4
P14-Full	Not solved	Not solved	
P17-Full	Not solved	Not solved	
P20-Full	11244	12913007	1
P20-Full	9396	16728930	2
P20-Full	10494	16768636	3
P20-Full	10494	10978125	4

Table 4: CNF encodings generated by SatPlan

Since full versions of some planning instances turned out to be too complex, 2 or 3 step versions were manually engineered to get better insights to why the full version was not solvable. These simpler versions include only atoms needed by the initial 2 or 3 reactions extracted from the full problem. Figure 3 displays the number of oxygen, hydrogen, carbon atoms and the total number of objects in each of the 2-3 step sub-problems. All 3 encoders, SASE, SatPlan and Madagascar were able to find a CNF encoding for problem p6-3 (3 step version of p6) and problem p20-2 (2 step version of p20). These encodings are promising since they were generated very quickly (under 0.1 seconds). Moreover, the number of variables and the number of clauses were also small as shown in Figure 4 which is based on the raw data from Table 5. Here, and subsequently, we only show encoding 1 for SatPlan since the other 3 encodings were very similar in both number of clauses and number of variables.

Splitting Domain then Encoding as CNF

Domain splitting was another option that we decided to explore. We decided to take all the actions needed for the problems mentioned in Table 1 and use action schema splitting mentioned in (Areces et al. 2014) to create a modified domain with $\gamma = 0.4$. This process splits an action with n parameters into many sub-actions. The minimum number of parameters for the sub-action is 2 when $\gamma = 0$ and n when

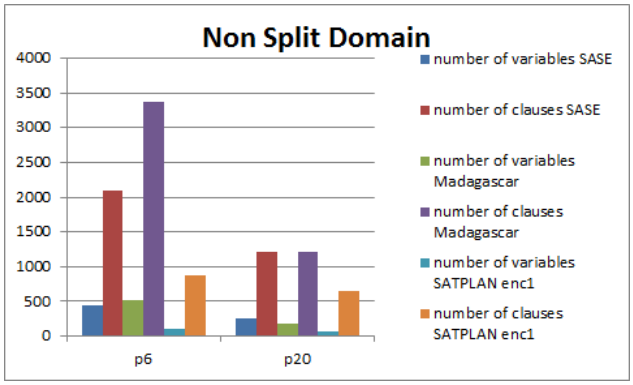


Figure 4: NoV and NoC for p6-3 and p20-2

	P6-3	P20-2
SASE, NoV	445	264
SASE, NoC	2102	1221
Madagascar, NoV	511	176
Madagascar, NoC	3364	1212
SatPlan enc1, NoV	105	70
SatPlan enc1, NoC	866	652

Table 5: Metrics for encodings of 2-3 step problems

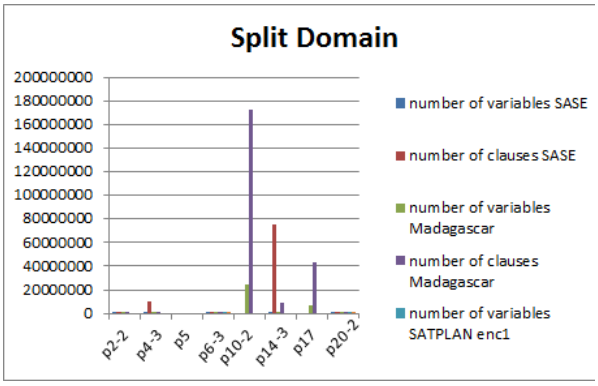


Figure 5: Metrics for encoding split domains

	p2-2	p4-3	p6-3	p10-2	p14-3	p17-full	p20-2
SASE (NoV)	25166	221510	3145	1146178	7634		
SASE (NoC)	421664	10016148	15179	75020254	67342		
Madagascar	5112	45200	2756	24399420	1116294	7322944	598
Madagascar	79907	495665	24436	172364287	8671435	43172172	6593
SatPlan enc1	862	499					
SatPlan enc1	5710	38589					

Table 6: The odd line is NoV and the even line is NoC

$\gamma = 1$. According to (Areces et al. 2014), the advantage of this process is an exponential reduction of the number of ground actions when actions are instantiated. This also produced interesting results. The participating programs were able to generate CNF encodings for more problems. However, the encodings were very large and it took several hours in some cases to generate them. Figure 5 shows a graph based on the raw data for simplified 2-3 step problems from Table 6. The empty entries mean the problem could not be encoded within 128GB of RAM. When comparing the results for problems 6-3step and p20-2step, it is clear that the number of variables was much larger for the split domain with $\gamma = 0.4$ than it was for the un-split domain. Similarly, the number of clauses produced by the encoders is much larger when using the split domain with $\gamma = 0.4$. Apparently, the CNF encodings from the split domain had been larger, since each action was split into multiple sub-actions.

The only full problem that was actually solved by all 3 planners, SASE, Madagascar and SatPlan, was problem 6 (both with the split and unsplit domains). Note that the other full problems discussed above were only encoded as CNF.

Table 7 shows the number of variables and number of clauses in CNF generated when encoding full p6 by SASE, SatPlan and Madagascar. When comparing the data from Table 7 with Table 3 and Table 4, it is clear how much larger are the encodings produced for the exact same problem. Note that both tables are based on a domain containing all 24 actions from Table 1. The only difference is that in Table 7 the encodings were generated based on the split domain with $\gamma = 0.4$ while in Tables 3 and 4 the encodings were based on the un-split domain. The last column in Table 7 includes the total time it took for the planners to encode and then to actually compute a plan. Observe that Madagascar is faster thanks to its specialized SAT solver. The program from (Heifets and Jurisica 2012) took 31 seconds to solve p6-full on an IBM supercomputer. No other full problems could be encoded using the split domain.

CNF encodings	NoV	NoC	Time (sec)
SASE	54881	490548	12.441
Madagascar	30660	382723	0.829
SATPLAN enc1	25348	2062809	29.846

Table 7: Problem 6-full with the full split domain $\gamma = 0.4$

This set of our experiments led us to several hypotheses. First, encoding split domains as CNF is not beneficial, since CNF encodings of un-split domains are more compact. Second, for un-split domains, the evaluated planners based on reducing planning to SAT can encode only the simplest planning instances. There is little indication they can scale up to solve all of the benchmark problems. Apparently, the number of objects in benchmark problems has an adverse effect on the abilities of SAT-based planners to scale up. Only p6-full with the lowest number (32) of atoms could be solved.

4.2 Fast Downward on the Split Domains

The split domain did however improve FD's results when running over the same problems. Previously, when we were debugging benchmark problems, FD took many hours on smaller unsplit domains that contained only actions needed to solve the problem. The domains used here have been preprocessed so that they have the $distinct(?x, ?y)$ predicate instead of negation of equality $(not\ (= ?x\ ?y))$ since splitting software requires STRIPS domains before it can run. In this set of experiments, we run a version of FD released

on October 9, 2015. FD was doing lazy greedy best-first search with FF heuristic. Table 8 shows the total **T**ime (in seconds) and peak **M**emory (in KB) that FD used to solve the problems using split smaller domains extracted manually for each problem from the larger split domain. For example, the sub-domain used for the 2 step version of p2 consisted of only the 2 actions needed to solve this simplified problem.

Problem	T $\gamma=0.4$	M $\gamma=0.4$	T $\gamma=0.0$	M $\gamma=0.0$
2-2Step	0.0616	5180	0.072	5616
4-3Step	0.6739	31448	0.357	13684
6-3Step	0.0061	3348	0.006	3340
10-2Step	58.6161	538952	0.867	34928
14-3Step	10.1912	62720	0.989	16956
20-2Step	0.0109	3476	0.011	3472

Table 8: FD's Time and Memory results when using smaller split sub-domains with $\gamma=0.4$, $\gamma=0.0$

The results in Table 8 demonstrate that for smaller domains and short, 2-3 step, planning problems splitting was effective and the finer split domains ($\gamma=0.0$) consumed less time and memory than the coarser split domain ($\gamma=0.4$).

To investigate the full problems, we decided to look carefully into the full domains using different γ values to see if different values improved the time it takes to solve a problem. We produced 2 more split domains using $\gamma=0.0$ and $\gamma=0.2$, in addition to the domain with $\gamma=0.4$ that we had before. The original full unsplit domain has 24 action, while the split domains with a gamma values of 0.0, 0.2 and 0.4 have 424, 336 and 228 actions, respectively. To see whether splitting would improve the time it took for a problem to be solved by FD, we obtained the following

data for the full problems 4,6 and 14 based on the full split domains.

P4	Total Time (Sec)	Peak Memory(KB)
non-split domain	Not Solved	Not Solved
split domain $\gamma=0.4$	Not Solved	Not Solved
split domain $\gamma=0.2$	45.7	78112
split domain $\gamma=0.0$	18.4	62521
P6	Total Time (Sec)	Peak Memory(KB)
non-split domain	49.3	275322
split domain $\gamma=0.4$	42.5	223080
split domain $\gamma=0.2$	39.1	115232
split domain $\gamma=0.0$	34.9	74256
P14	Total Time (Sec)	Peak Memory(KB)
non-split domain	Not Solved	Not Solved
split domain $\gamma=0.4$	Not Solved	Not Solved
split domain $\gamma=0.2$	22546.4	2262432
split domain $\gamma=0.0$	4306.6	2486320

Table 9: FD: problems 4,6 and 14 split/unsplit full domains (domains containing 24 actions)

Table 9 shows some advantages of the finer split domains, since problems 4, 6 and 14 took less time for smaller γ values. However, the other full problems, i.e. 2,3,5,10,17 and 20 were not solved for any of the full split domains. All these problems need more than 128GB of RAM. Therefore, splitting was not completely successful even for $\gamma=0.0$.

The advantages of splitting were not always demonstrated. P20 (full, 11-step version) was not solved by FD with neither the split nor the unsplit full domains due to its complexity. We decided to create 2 smaller sub-domains, where only the actions needed for problem 20 are present, one based on the split domain with a $\gamma=0.4$ and one based on the split domain with a $\gamma=0.0$. Table 10 shows the time, memory and the number of initial candidates FD produced.

P20	Total Time (Sec)	Peak Memory(KB)	Initial Candidates
non-split	1.31	23368	6
$\gamma=0.4$	2750	3698780	74
$\gamma=0.0$	38.8	80076	93

Table 10: Smaller split and unsplit sub-domains: p20

The number of initial candidates is a metric that is produced by FD based on the domain and problem given. The split domain has many more initial candidates that FD needs to explore than the unsplit domain. This is the reason why FD takes more time and needs more memory to find the answer when solving the long planning problem using the split domain. As you can see in Table 10, FD needs to only check 6 initial candidates when using the unsplit domain which consumes less time and memory. Also, for $\gamma=0.4$, the domain has more initial candidates than the unsplit domain, but more complex (more parameters) reactions than for $\gamma=0.0$, so it is has the worst of both worlds. It is mixing the worst things of the other two cases into one. Notice also that in the split domain the length of the plan is much longer than in the un-split domain because a greater number of sub-actions should be executed sequentially. Therefore, in the split domain, finding a plan can take more time. Overall, splitting increases the number of initial candidates, but simplifies the actions. This example demonstrates shows the limitations of splitting in some situations.

Simplifying the domain to improve time and memory usage. It was interesting to investigate how much performance of FD would improve, if a few atoms are removed from the domain reactions thereby alleviating in part the grounding problem. In some reactions, there are hydrogen atoms that remain invariant, i.e, the reactions have no effects on their bonds with carbon atoms. Since they do not really participate in reactions, removing a few of them is harmless, in a sense that the planning instances still should be solvable after doing this minor modification. The data that we collected in Table 11 show that by removing hydrogen atoms and their bonds with carbons, the time and memory improved, but with varying efficiency.

P4	Total Time (Sec)	Peak Memory(KB)
FullDomain $\gamma=0.0$	18.4	62521
Domain $\gamma=0.0$ with 1 H removed	16.7	40632
P6		
FullDomain $\gamma=0.0$	34.9	74256
Domain $\gamma=0.0$ with 1 H removed	33	72772
Domain $\gamma=0.0$ with 2 H removed	16.7	39528
P14		
FullDomain $\gamma=0.0$	4306.6	2486320
Domain $\gamma=0.0$ with 1 H removed	773	513524

Table 11: Removing hydrogens to improve time/memory

Heuristic	P4	P6	P14
eager_greedy, ff	95 s	82 s	36.4 h
lazy_greedy, add	33.8 s	19 s	69946 s
lazy_greedy, ipdb	–	–	–
lazy_greedy, lmcount	–	163 s	–
lazy_greedy, max	–	–	–
lazy_greedy, merge&shrink	–	–	–
astar, add	37 s	19 s	73591 s
astar, ipdb	–	–	–
astar, lmcount	–	3.2 h	–
astar, max	–	–	–
astar, merge&shrink	–	–	–

Table 12: Time for P4, P6 and P14 using different search

Heuristic	P2-2	P6-3	P10-2	P14-3	P20-2
eager_greedy, ff	0.12	0.52	0.998	1.05	0.0117
lazy_greedy, add	0.08	0.02	0.93	1.04	0.010
lazy_greedy, ipdb	–	0.052	–	–	–
lazy_greedy, lmcount	–	0.10	–	–	0.071
lazy_greedy, max	–	–	–	–	–
lazy_greedy, m&s	–	11.7	–	–	–
astar, add	0.09	0.4	0.9	1.2	0.011
astar, ipdb	–	0.05	–	–	–
astar, lmcount	–	0.19	–	–	0.08
astar, max	–	–	–	–	–
astar, m&s	–	–	–	–	–

Table 13: Time (sec) for 2-3 step problems per heuristic

Heuristics All data generated by FD above are collected from the lazy_greedy(ff) search. We experimented with a few different heuristics and search combinations using the finest split domain ($\gamma = 0.0$). As you can see in Table 12, eager_greedy(ff) was only able to solve problems 4 and 6 in a reasonable time but took over 36 hours to solve problem 14. The astar(add) optimal search performed well: FD solved problems 4, 6, 14 in 37, 19, 73591 seconds, respectively. However, it was still slower than lazy greedy with ff: see Table 9 above. The A^* search with ipdb, max and the merge and shrink heuristics produced negative results; "–" means FD was not able to solve the problem in a reasonable time. FD doing A^* search with lmcount took over 3 hours to solve problem 6, and was not able to solve problems 4 and 14. In comparison, lazy_greedy performed better. For example, when using lmcount as the heuristic, P6 was solved in 163 seconds, which is a huge improvement in time when compared to 3.2 hours for A^*. A similar pattern can be seen for the 2-3 step problems: see Table 13. Only eager greedy with ff and astar(add) were able to solve the 2-3 step problems, but lazy greedy with ff did the best when compared with the other options. These data are somewhat preliminary, but they show that this new domain can serve as a challenge for researchers developing heuristics.

5 Gamer and L-RPG

Gamer (Kissmann and Edelkamp 2011; Kissmann 2012) is a symbolic planner based on binary decision diagrams. Unfortunately, it was not possible to build Gamer on a 64 bit machine. However, Gamer did run on a 32 bit machine with 4GB of RAM. It was able to solve only P2-2step, P4-3step and P6-3step using sub-domains containing only the actions needed for the problems. The actions in the domains were split with $\gamma = 0.0$ for best results. The same 3 problems were not solved when a non split domain was used. Furthermore, none of the full problems were solved using split and non split domains. It is possible that the reason for not solving full problems is the limited available RAM. However, the fact that the 2-3step problems were not solved when using the non split domain does not support that.

L-RPG (Ridder 2015; 2014; Ridder and Fox 2014) is a forward-chaining planner that does not rely on grounding. The planner was developed as a solution to memory constraints issues that other state-of-the-art planners have. L-RPG looks like a promising solution to the benchmark problems. However, it was not possible to solve any of the planning instances due to bugs found in the code of L-RPG.

6 Conclusion and Future Work

FD's results for problems 4, 6 and 14 are somewhat comparable in terms of time with data reported in Table 1 from (Heifets and Jurisica 2012), despite differences in hardware. FD was able to solve problems 4 and 6 using the finest split domain slightly slower than the proof-number search on an IBM supercomputer that took 15sec and 31sec, respectively. However, FD was able to solve problem 14 faster using the split domain ($\gamma = 0.0$); the proof number search on an IBM super-computer took 5138sec vs FD's 4306sec as reported in our Table 9 above. However, FD could not solve problems 2,3,5,10 within 128GB of memory even using split domains.

We can see from the results of the experiments that encoding the problems as CNF did not generate good results. The produced encodings were very large due to grounding. However, we have demonstrated that actions schema splitting transformation is practically useful since splitting did improve FD's results when compared to unsplit domains, at least for some problems. At the same time, splitting helped only with solving a few of the full benchmark problems. Other ideas should be explored before a planner can reliably compete with undergraduate students in solving Heifets benchmark problems. The benchmark (Heifets 2012) remains a serious challenge for modern AI planners.

There are several future work possibilities. The research in LRPG (Ridder 2015) can help because lifted planning can be promising with the proposed domain. Similarly, lifted encodings (Kautz, McAllester, and Selman 1996) should be explored. Moreover, reducing planning to QBF (Cashmore, Fox, and Giunchiglia 2013; Cashmore 2013) may be beneficial for this planning domain.

7 Acknowledgement

R.M. (the first author) would like to thank the Undergraduate Program of the Computer Science Department at Ryerson University for providing the opportunity to take part in the undergraduate thesis course and for providing access to a cloud based Linux virtual machine that has been allocated 128GB of RAM. Thanks to Dr. Anne Johnson (Dept. of Chemistry and Biology, Ryerson) for correcting several chemical errors in the reactions code.

References

Accelrys. 2011. *CTfile Formats*. Accelrys. http://download.accelrys.com/freeware/.

Areces, C.; Bustos, F.; Dominguez, M. A.; and Hoffmann, J. 2014. Optimizing planning domains by automatic action schema splitting. In Chien et al. (2014).

Bøgevig, A.; Federsel, H.-J.; Huerta, F.; Hutchings, M. G.; Kraut, H.; Langer, T.; Löw, P.; Oppawsky, C.; Rein, T.; and Saller, H. 2015. Route design in the 21st century: The IC-SYNTH software tool as an idea generator for synthesis prediction. *Organic Process Res. & Developm.* 19(2):357–368.

Cashmore, M.; Fox, M.; and Giunchiglia, E. 2013. Partially grounded planning as quantified boolean formula. In Borrajo, D.; Kambhampati, S.; Oddi, A.; and Fratini, S., eds., *Proceedings of the Twenty-Third International Conference on Automated Planning and Scheduling, ICAPS 2013, Rome, Italy, June 10-14, 2013*. AAAI.

Cashmore, M. 2013. *Planning as Quantified Boolean Formulae*. Ph.D. Dissertation, University of Strathclyde, Dept. of Computer and Information Sciences.

ChemAxon. 2015. JChem software. http://www.chemaxon.com.

Chien, S. A.; Do, M. B.; Fern, A.; and Ruml, W., eds. 2014. *Proceedings of the Twenty-Fourth International Conference on Automated Planning and Scheduling, ICAPS 2014, Portsmouth, New Hampshire, USA, June 21-26, 2014*. AAAI.

Cook, A.; Johnson, A. P.; Law, J.; Mirzazadeh, M.; Ravitz, O.; and Simon, A. 2012. Computer-aided synthesis design: 40 years on. *Wiley Interdisciplinary Reviews: Computational Molecular Science* 2(1):79–107.

Edelkamp, S., and Hoffmann, J. 2004. PDDL2.2: The Language for the Classical Part of the 4th Intern. Planning Competition. Technical Report 195, Universität Freiburg, Institut für Informatik.

Ernst, M.; Millstein, T.; and Weld, D. 1997. Automatic satcompilation of planning problems. In *Proceedings of the Fifteenth International Joint Conference on Artificial Intelligence, IJCAI 97, Nagoya, Japan, August 23-29, 1997, 2 Volumes*, 1169–1177.

Heifets, A., and Jurisica, I. 2012. Construction of new medicines via game proof search. In Hoffmann, J., and Selman, B., eds., *AAAI*, 1564–1570. AAAI Press.

Heifets, A. 2012. *Benchmark problems, 2012*. Available at http://www.cs.toronto.edu/~aheifets/ChemicalPlanning/.

Helmert, M., and et al. 2015. *The Fast Downward Planning System*. http://www.fast-downward.org/ .

Huang, R.; Chen, Y.; and Zhang, W. 2010. A novel transition based encoding scheme for planning as satisfiability. In *Proceedings of the Twenty-Fourth AAAI Conference on Artificial Intelligence, AAAI 2010, Atlanta, Georgia, USA, July 11-15, 2010*.

James, C.; Weininger, D.; and Delany, J. 2011. *Daylight Theory Manual Ver. 4.9 (08/01/11)*. http://www.daylight.com/dayhtml/doc/theory/index.html .

Judson, P. 2009. *Knowledge-Based Expert Systems in Chemistry: Not Counting on Computers*. RSC Theoretical and Computational Chemistry. Royal Society of Chemistry.

Kautz, H. A., and Selman, B. 1992. Planning as satisfiability. In *ECAI*, 359–363.

Kautz, H. A.; McAllester, D. A.; and Selman, B. 1996. Encoding plans in propositional logic. In *Proceedings of the Fifth International Conference on Principles of Knowledge Representation and Reasoning (KR'96), Cambridge, Massachusetts, USA, November 5-8, 1996.*, 374–384.

Kautz, H.; Selman, B.; and Hoffmann, J. 2006. SATPLAN: Planning as Satisfiability. In *Abstracts of the 5th International Planning Competition, 2006*. http://www.cs.rochester.edu/users/faculty/kautz/satplan/ .

Kissmann, P., and Edelkamp, S. 2011. Gamer, a general game playing agent. *KI* 25(1):49–52.

Kissmann, P. 2012. *Symbolic Search in Planning and General Game Playing*. Ph.D. Dissertation, Universität Bremen, Germany. http://nbn-resolving.de/urn:nbn:de:gbv:46-00102863-15.

Masoumi, A.; Antoniazzi, M.; and Soutchanski, M. 2015. Modeling organic chemistry and planning organic synthesis. In *Proceedings of the 1st Global Conference on Artificial Intelligence (GCAI 2015), EasyChair Proceedings in Computing*, volume 36, 176–195.

Ravitz, O. 2013. Data-driven computer aided synthesis design. *Drug Discovery Today: Technologies* 10(3):e443 – e449. http://www.chemplanner.com.

Ridder, B., and Fox, M. 2014. Heuristic evaluation based on lifted relaxed planning graphs. In Chien et al. (2014).

Ridder, B. C. 2014. *Lifted Heuristics: Towards More Scalable Planning Systems*. Ph.D. Dissertation, Kings College, Department of Informatics, London, UK.

Ridder, B. 2015. *L-RPG: Introducing a Lifted Forward-Chaining Planner*. Available at https://www.assembla.com/spaces/MyPOP/subversion/source.

Rintanen, J. 2014. Madagascar: Scalable Planning with SAT; an overview of the techniques in the Madagascar (M, Mp, MpC) planners. In *International Planning Competition*. https://users.ics.aalto.fi/rintanen/papers/Rintanen14IPC.pdf .

Rintanen, J. 2015. *Madagascar software*. http://users.ics.aalto.fi/rintanen/MADAGASCAR.TAR .

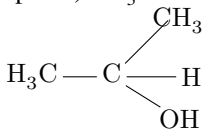
Szymkuć, S.; Gajewska, E. P.; Klucznik, T.; Molga, K.; Dittwald, P.; Startek, M.; Bajczyk, M.; and Grzybowski, B. A. 2016. Computer-assisted synthetic planning: The end of the beginning. *Angewandte Chemie International Edition* 55(20):5904–5937.

Appendix

In organic chemistry, many molecules are compounds of bonded carbon and hydrogen atoms and each carbon atom has the valence 4. To simplify the figures, it is usual to assume that all unnamed nodes are carbon atoms with the appropriate number of hydrogen atoms attached by default.

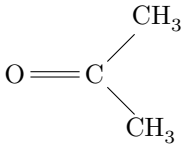
To explain what happens in p5, we show main molecules participating in the solution to p5. We skip here PCC (two

of them are present in the initial state of p5) and other small molecules since they have been already discussed in Preliminaries section. In the initial state of p5, one of the molecules participating in oxidation reaction is isopropyl alcohol[2] (isopropanol) $CH_3CH{-}OH{-}CH_3$

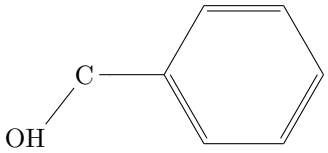


Isopropyl alcohol

The product of the first reaction is acetone $(CH_3)_2CO$, or propanone, a compound used in many households.[3]



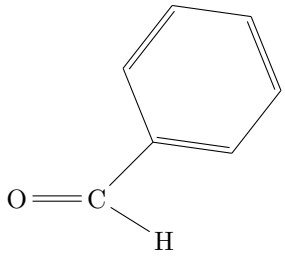
Acetone



Benzyl alcohol

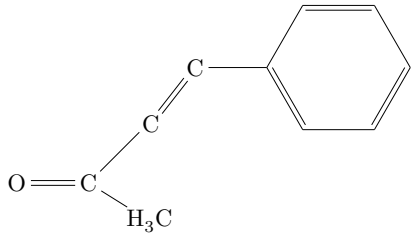
The second oxidation reaction takes benzyl alcohol (known also as phenylmethanol) molecule $C_6H_5CH_2OH$ and produces a benzaldehyde[4] molecule C_6H_5CHO. This molecule is called benzyl alcohol because it has a hydroxyl group $-OH$ attached to the carbon atom that also has a single bond with the phenyl ring. The phenyl ring can be viewed as a benzene ring, minus a hydrogen. It is often called aromatic ring because all (the assumed, present by default) carbons around the ring have aromatic bonds with each other. It is common to draw this aromatic ring with a sequence of alternating single and double bonds, while in reality they are aromatic bonds. The carbon attached to the hydroxyl group has two other assumed single bonds with hydrogen atoms present by default.

The name benzaldehyde can be explained by the presence of an aromatic phenyl ring that is attached to a carbon atom. This carbon has a single bond with a hydrogen atom, and a double bond with an oxygen atom thereby forming the carbonyl group C=O. Carbonyl containing compounds with the bond to hydrogen are known as aldehyde. Benzaldehyde is simplest aromatic aldehyde; it is industrially useful.



Benzaldehyde

The last 3rd reaction, $aldolCondensation$ reaction,[5] takes acetone, which is a simplest ketone, takes benzaldehyde, which is an aldehyde, takes sodium hydroxide NaOH, also known as lye and caustic soda, and produces the target molecule, named 4-phenylbut-3-en-2-one, also known as benzalacetone, benzylideneacetone,[6] which is a ketone.



Benzylideneacetone

The following PDDL code provides detailed and complete description of this last reaction.

```
(:action aldolCondensation
    :parameters (?o_10 - oxygen ?o_4 - oxygen
?h_11 - hydrogen ?na_14 - sodium ?c_1 - carbon
?c_5 - carbon ?o_15 - oxygen ?h_8 - hydrogen
?h_9 - hydrogen  ?h_12 - hydrogen
?h_13 - hydrogen ?r1_2 - object
?r1_3 - object ?o_7 - oxygen ?c_6 - carbon)
    :precondition (and (not (= ?c_1 ?c_5))
 (not (= ?h_11 ?h_8)) (not (= ?h_11 ?h_9))
 (not (= ?h_8 ?h_9)) (not (= ?o_10 ?o_4))
 (not (= ?o_10 ?o_15)) (not (= ?o_4 ?o_15))
(not (= ?h_11 ?h_12)) (bond ?o_10 ?h_11)
 (bond ?o_10 ?h_12) (bond ?na_14 ?o_15)
(bond ?h_13 ?o_15) (not (= ?r1_2 ?r1_3))
 (bond ?r1_2 ?c_1) (bond ?r1_3 ?c_1)
(doublebond ?o_4 ?c_1) (not (= ?h_9 ?h_8))
 (not (= ?c_6 ?c_5))
(bond ?c_6 ?c_5) (bond ?h_9 ?c_5)
(bond ?h_8 ?c_5) (doublebond ?o_7 ?c_6))
    :effect (and (not (bond ?o_10 ?h_11))
(not (bond ?h_11 ?o_10)) (bond ?o_10 ?h_8)
 (bond ?h_8 ?o_10) (bond ?h_11 ?o_4)
(bond ?o_4 ?h_11) (not (bond ?na_14 ?o_15))
(not (bond ?o_15 ?na_14)) (bond ?na_14 ?o_4)
 (bond ?o_4 ?na_14) (bond ?h_9 ?o_15)
(bond ?o_15 ?h_9) (not (doublebond ?o_4 ?c_1))
 (not (doublebond ?c_1 ?o_4))
 (doublebond ?c_1 ?c_5)(doublebond ?c_5 ?c_1)
 (not (bond ?h_8 ?c_5))(not (bond ?c_5 ?h_8))
 (not (bond ?h_9 ?c_5))(not (bond ?c_5 ?h_9) )
)
)
```

[2]https://en.wikipedia.org/wiki/Isopropyl_alcohol

[3]https://en.wikipedia.org/wiki/Acetone

[4]https://en.wikipedia.org/wiki/Benzaldehyde

[5]https://en.wikipedia.org/wiki/Aldol_condensation

[6]https://en.wikipedia.org/wiki/Benzylideneacetone