Towards a Planning-Based Approach to the Automated Design of Chemical Processes

Arman Masoumi[1], Andrea Marrella[2], and Mikhail Soutchanski[1]

[1] Ryerson University, Toronto, Ontario, Canada
[2] Sapienza Università di Roma, Rome, Italy

Abstract. The design of chemical processes is a central problem in organic chemistry. A chemical process is a sequence of chemical reactions capable of producing a target compound from some starting stage molecules. The manual specification of such processes can be time-consuming and error-prone, due to the high number of reactions involved and their complex chemical conditions. To tackle this issue, we propose a planning-based approach and a framework for the automated design of chemical processes. Specifically, we argue that this problem can be reduced to a planning problem in Artificial Intelligence. To this end, we adapt the situation calculus and PDDL to the task of modeling molecules and capturing semantics of generic chemical reactions, and we conduct experimental study including empirical assessment of a PROLOG planner and two state-of-the-art planners on a set of benchmark problems.

1 Introduction

Business Process Management [22] (BPM) is a very active research area, because it is highly relevant from a practical point of view while at the same time it offers many challenges for researchers. BPM is based on the observation that each product that a company provides to the market is the outcome of a number of activities. *Business processes* are the key instruments to organizing these activities and to improving our understanding of their interrelationships. Traditionally, BPM has focused on the support of predictable and repetitive business processes, that include mainly production and administrative processes (such as manifacturing, order handling, etc.) [12]. In recent years, the maturity of process management methodologies has led to the application of process-oriented approaches in new knowledge-intensive scenarios [6], such as healthcare [19] and emergency management [15]. In a knowledge-intensive process, the sequence of activities depends heavily on the specifics of the context (e.g., which resources and what particular options exist at the time of the process definition). Such processes do not have the same level of repeatability as classical business processes, and their instances may change on a case-by-case basis, depending on the context [19].

An interesting and challenging example comes from the field of *Organic Chemistry*, where the design of *chemical processes* is central for the generation of complex chemical compounds starting from natural raw materials. It enables chemists to discover new drugs by producing compounds that do not form naturally. Basically, a chemical process consists of a sequence of *chemical reactions*, that lead to the transformation of one set of chemical substances to another. Chemical substances are made up of atoms of different elements, joined together by chemical bonds. A chemical reaction is characterized by a chemical change that produces one or more final substances, by involving the breaking of existing bonds and the formation of new ones. The correct execution of a chemical process aims at generating a complex compound by involving several chemical reactions to be enacted in sequence. However, to identify which reactions are capable of producing the target compound is a complex activity to be handled manually, even for the most skilled chemists. In fact, each reaction, for being enacted, requires that certain chemical conditions are satisfied at a specific point of the chain, possibly when other reactions have already been performed and further substances have been produced.

To tackle this issue, building on our previous work [16], we propose a planning-based approach and a framework for the automated design of chemical processes. Specifically, we allow a chemist to select a set of starting molecules from a specific library and to establish the target compound to achieve, and we leverage on planning techniques developed in Artificial Intelligence (AI) for generating a chemical process that can transform the starting molecules into the target ones. This problem is also referred as the CAOS (*Computer-Assisted Organic Synthesis*) problem [3]. We show that the CAOS problem can be conveniently formulated as a planning problem in *Situation Calculus* [20], a well-known logical formalism used in AI for representing and reasoning on actions, and in PDDL (the *Planning Domain Definition Language* [7]). Specifically, we adapt the situation calculus and PDDL to the task of modeling molecules and capturing semantics of generic chemical reactions, and we formulate the process design problem as a planning problem in AI. We investigate two different computational approaches to solving this kind of planning problem. One of the planners that we conducted research solves the planning problem directly in situation calculus and accepts PROLOG encoded specification of the problem. This planner can take advantage of domain-specific declarative heuristics built ad-hoc for reducing significantly the search space [16]. Then, we solve the same planning problem with two state-of-the-art planners working with a PDDL input and domain-independent heuristics. Finally, we present a comparative analysis of the performance of the tested planners in solving different process design problems.

2 Case Study

In this paper, we focus on *organic chemistry*, a discipline dedicated to studying molecules involving carbon-hydrogen bonds and their reactions. *Molecules* are formed by bonded *atoms*. It is common to represent atoms with their shorthand (e.g., the shorthand for carbon is C) from the periodic table as vertices of a graph, and the chemical bonds as edges of the graph connecting the atoms. The molecules presented in this paper are formed using *single bonds* and *double bonds*. A single/double bond is represented as a single/double edge between atoms. The process of transforming one or more molecules to a different set of molecules is referred to as a *chemical reaction*. A reaction is described with a *chemical equation*, which graphically presents the molecules before the reaction, known as *substrates*, and the new molecules that are created after the reaction, called the *products*. If a molecule is unchanged before and after a reaction, but has been necessary for the reaction to happen, then it is termed a *catalyst*. What changes in a chemical reaction is essentially the bonds between the atoms of the molecules in the substrate. Representing *generic chemical reactions* requires representing and identifying *classes of molecules*. Molecules within the same chemical class have similar chemical characteristics. For example, *alkanes*, *alcohols* and *esters* are well-known chemical classes that display unique behaviors in reactions. A few generic chemical reactions are in shown in Fig. 1, where R and R′ are *alkyls*, i.e., chemical compounds that consist solely of acyclic single bonded carbon and hydrogen atoms with the generic formula C_nH_{2n+1}. For example, *methyl* CH_3- and *ethyl* CH_3-CH_2- are alkyls. Specifically:

- The reaction between *ester* and *water* results in a *carboxylic acid* and *alcohol*, cf. Fig. 1(a). This reaction needs a strong acid as a catalyzer.
- The reaction between an *alcohol* and *strong base* NaH results in an *alkoxide salt* and *hydrogen* molecule H_2, cf. Fig. 1(b).
- The reaction between *alkoxide salt* and an *alkyl halide* results in an *ether* and a *salt*, cf. Fig. 1(c). Here, X is any *halogen*, and Y is any *alkali metal*. Halogens and alkali metals are specific groups of atoms. For example, Fluorine F and Chlorine Cl are halogens, and Sodium Na is an alkali metal.
- The reaction between *ether* and *mineral acid* HCl results in *alcohol* and *alkyl halide*, cf. Fig. 1(d).

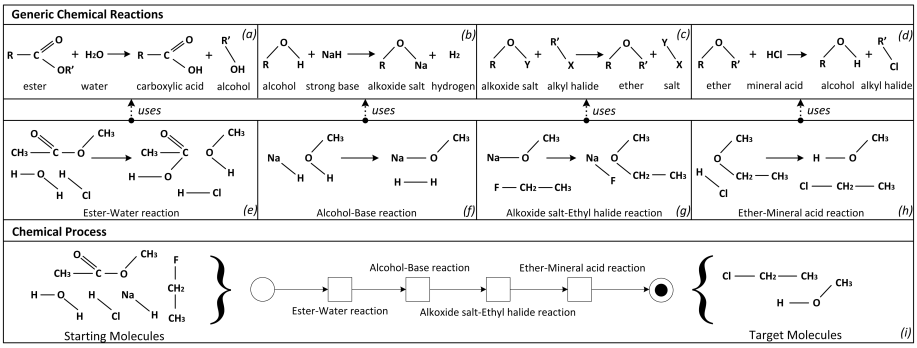

Fig. 1. The list of generic/specific reactions used for designing a chemical process.

In organic chemistry, a relevant problem consists of designing *chemical processes*, i.e., chains of chemical reactions that produce new molecules (i.e., a target compound) using some starting stage molecules. Most often, there is no one reaction that can transform the starting molecules to the desired goal molecules, rather a sequence of different reactions is needed. We consider a simple case study that applies the above reaction schemas for designing a chemical process that requires a sequence of four reactions. Our goal is to synthesize an *alcohol* molecule and an *ethyl chloride*, starting from methyl acetate $CH_3{-}COO{-}CH_3$, water H_2O, hydrochloric acid HCl, sodium hydride NaH and ethyl fluoride $CH_3{-}CH_2{-}F$. The goal molecule can be synthesized as follows:

- First, the *methyl acetate* (an ester), the *water molecule*, and the *hydrochloric acid* react to produce *methanol* (an alcohol) and *acetic acid* (a carboxylic acid). Notice that *hydrochloric acid* HCl is the catalyst in this reaction, cf. Fig. 1(e).
- Second, the *methanol* (an alcohol) reacts with the *sodium hydride* (a strong base) to produce *sodium methoxide* (an alkoxide salt) and H_2, cf. Fig. 1(f).
- Third, the *sodium methoxide* (an alkoxide salt) reacts with *ethyl fluoride* (an ethyl halide) to produce *methyl ethyl ether* (an ether) $CH_2{-}O{-}C_2H_5$, cf. Fig. 1(g).
- Lastly, the *methyl ethyl ether* reacts with the *hydrochloric acid* (a mineral acid) to produce *methanol* (an alcohol) and *ethyl chloride* (an alkyl halide), cf. Fig. 1(h).

Fig. 1(i) shows the chemical process required for solving our synthesis problem. The process is represented in BPMN [2] (Business Process Modeling Notation), a diagramming language designed to specify a process in a standardized way. Intuitively, given a set of starting molecules, an execution of the process starts at ○ and ends at ⊙; an *activity* □ reflects a chemical reaction executed by the process; *transitions* are binary relations describing in which order the reactions have to be performed. An execution of the chemical process produces a set of target molecules, as shown in Fig. 1(i). In order to find manually the sequence of reactions involved in a chemical process, a chemist needs to first identify the chemical classes of each molecule at hand, and then refer to her/his knowledge of chemical reactions to find out which of these molecules can react with each other, and what would the product be if they reacted. The next task is to identify whether the hypothetical product is useful, and whether that product can react with some other molecules at hand to produce yet another useful product and so on. These tasks needs to be repeated until the goal molecule is synthesized, or when the chemist realizes there is no known way of synthesizing the goal molecule from the molecules at hand. The problem becomes more intricate when the number of molecules is large, when the molecules are complex, and when a large number of reactions is needed before the goal molecule is synthesized. In practice, even the most skillful chemists cannot solve synthesis problems exceeding from a small number of steps manually.

3 Preliminaries

Situation Calculus. The *situation calculus* (SC) is a logic language designed for representing and reasoning about dynamic domains [20]. In SC, a dynamic world is modeled as progressing through a series of *situations* as a result of various *actions* being performed. A *situation* s is a first-order term denoting the sequence of actions performed so far. A special binary function symbol $do(a, s)$ denotes the next situation resulting from the performance of action a in situation s. The special constant S_0 stands for the *initial situation*, namely the empty action sequence. An *Initial Theory* D_{S_0} represents the set of axioms in S_0, before any action has occurred.

Conditions whose truth value may change are modeled by means of *fluents*. Technically, these are predicates taking a situation term as their last argument. Fluents may be thought of as "properties" of the world whose values may vary across situations. Changes in fluents (resulting from executing actions) are specified through a set D_{ss} of *successor state axioms SSAs*. In particular for each fluent F we have a SSA as follows: $F(\overrightarrow{x}, do(a, s)) \Leftrightarrow \Gamma_F(\overrightarrow{x}, a, s)$, where $\Gamma_F(\overrightarrow{x}, a, s)$ is a formula with free variables fully capturing the truth-value of fluent F on a tuple of objects $\overrightarrow{x}$ when action a is performed in situation s. Besides successor state axioms, SC is characterized by a set D_{pa} of *action precondition axioms PAs*, which specify whether a certain action is executable in a situation. PAs have the form: $Poss(a, s) \Leftrightarrow \Pi_a(s)$, where the formula $\Pi_a(s)$ defines the conditions under which the action a may be performed in the situation s.

A basic action theory (BAT) $D = D_{S_0} \cup D_{pa} \cup D_{ss}$ is the set of the relevant axioms that is used to model actions and their effects in SC. BATs might also be augmented with *abbreviations*. They are declared as predicates (with situation argument) defined by means of a formula uniform in s that can mention only other abbreviations or fluents at s. Abbreviations, unlike fluents, are not directly affected by actions. However, similarly to fluents, their truth value may vary from situation to situation.

Planning Algorithms. Planning systems are problem-solving algorithms that operate on explicit representations of states and actions [18]. The standard representation language of classical planners is known as the *Planning Domain Definition Language* [7] (PDDL); it allows one to formulate a problem PR through the description of the initial state of the world $init_{\mathsf{PR}}$ and of the desired goal condition $goal_{\mathsf{PR}}$. The domain PD of the planning problem mostly introduces relational predicates and a set of possible action definitions Ω. An *action schema* defines the condition under which an action $a \in \Omega$ can be executed, called *pre-conditions* Pre_a and its *effects* Eff_a on the state of the world. A planner that works on such inputs generates a sequence of actions (the *plan*) that leads from the initial state to a state meeting the goal. In this paper, we focus on *classical planning* techniques, characterized by fully observable, static, and deterministic domains, in which plans can be computed in advance and then applied unconditionally. Classical planning has made huge advances in the last twenty years, leading to solvers able to create plans with thousands of actions for problems described by hundreds of propositions. In this work, we represent planning domains and planning problems making use of PDDL 2.2 [7], that is characterized for enabling the representation of planning domains including operators with derived predicates.

4 The General Framework

One of the main obstacles in applying AI techniques to real problems is the difficulty to model the domains. Usually, this requires that people that have developed the AI system carry out the modeling phase, since the representation depends very much on a deep knowledge of the internal working of the AI tools. Since we are aware that a chemist that wants to automatically design a chemical process is probably not an expert of AI techniques, we worked on a framework that allows non-experts entering knowledge on molecules and chemical reactions through a user-friendly interface.

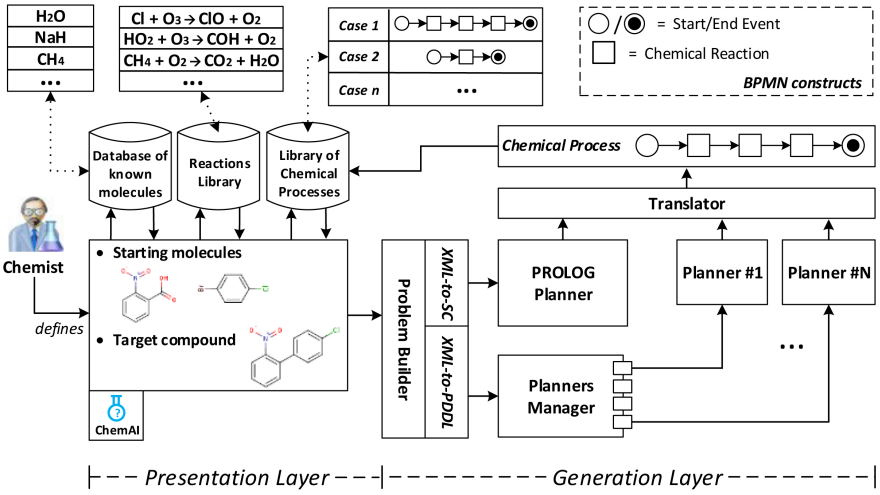

Fig. 2. The general framework for designing chemical processes.

Specifically, our approach for designing chemical processes relies on two main logical layers as shown in Fig. 2. The *Presentation Layer* has a twofold purpose. On one hand, it allows a chemist to visualize and quickly draw new molecules, to retrieve generic chemical reactions from a dedicated *Reactions Library* or to define completely new reactions. For these tasks we rely on *ChemAxon* products - `http://www.chemaxon.com/`, which provide free Java-based chemistry editors for drawing, retrieving and visualizing chemical structures. Every entry in the Reactions Library characterizes a reaction with a *(i)* name, *(ii)* a set of chemical conditions, that must be satisfied for making the reaction executable (e.g., the list of required *catalysts*) and *(iii)* the finite number of molecules involved in the reactions, i.e., the *substrates* and the *products*. On the other hand, we provide a GUI-based tool named *ChemAI*. It can be used by a chemist to specify a *chemical case*, i.e., a set of *starting molecules* (retrieved from a *Database of known molecules*) and a *target compound* to achieve. At this point, ChemAI interacts with a *Library of Chemical Processes* that have been built previously for specific chemical cases. If a chemical process exists for the current value of the chemical case, it is graphically returned to the chemist, by showing the chain of chemical reactions required for synthesizing the target compound. Otherwise, if no process exists for the current value of the chemical case, ChemAI first retrieves from the Reactions Library a set of generic reactions that conform with the chemical classes of the starting molecules (i.e., they can be possibly used for generating the target compound), then builds an XML file including the selected reactions and a representation of the chemical case, and finally passes these information to the Generation Layer.

The *Generation Layer* is in charge of automatically designing a chemical process whose execution leads from the starting molecules to the target compound. The *Problem Builder* component acts as unique entry point from the Presentation Layer. It provides two software modules, named *XML-to-SC* and *XML-to-PDDL*. They take in input the XML specification received from ChemAI and, respectively, translate it in SC/PDDL readable formats. The formalization of chemical reactions in SC BATs/PDDL requires to explicitly represent molecules, their atoms and the bonds between them. Let's focus on the translation algorithm implemented in *XML-to-SC*, that works in three separate steps. First, the algorithm identifies which atoms and which bonds between atoms are involved in the starting molecules. This knowledge is used for: *(i)* representing atoms with situation independent predicates. For example, $Carbon(C_1)$ means the constant C_1 represents a carbon atom; *(ii)* understanding which type of bond exists between the atoms in the starting molecules. We use the fluent $Bond(x, y, s)$ (respectively,

$DoubleBond(x, y, s)$) for describing if atom x has a single bond (respectively, a double bond) with atom y in the situation s. Secondly, the translation algorithm analyzes the selected chemical reactions and generates the sets D_{pa}, D_{ss} and the abbreviations required for reasoning over reactions in SC. Specifically, for each generic reaction X:

- a corresponding SC action a_x is created. For example, the reaction between alcohol and strong bases can be represented as a SC action $a\_b\_r(oxOfAlcohol, hydOfAlcohol, xOfBase, metalOfBase)$, cf. Fig. 1(f). Here, four atoms change bond, and they are listed as the arguments of the action;
- the substrates and the chemical conditions associated to X are used for generating a PA for a_x. A PA introduces the molecules on the left hand side of the reaction (i.e., the substrates and the possible catalysts). For example, the PA needed for executing the a_b_r reaction is $Poss(\, a\_b\_r(oxOfAlcohol, hydOfAlcohol, xOfBase, metalOfBase), s\,) \leftrightarrow$ $alcohol(oxOfAlcohol, hydOfAlcohol, s) \wedge base(xOfBase, metalOfBase, s)$.
- the substrates and the products associated to X are elaborated for inferring the SSA of a_x. It takes into account all the bonds that are formed and cleaved as the result of the reaction. For example, the effects of a_b_r are captured in the SSA: $Bond(x, y, do(a, s)) \leftrightarrow (\exists\, z, u)(\, a = a\_b\_r(x, z, u, y)) \vee (...)$ $\vee\, (Bond(x, y, s) \wedge (\neg\exists\, u, z)(\, a = a\_b\_r(x, y, u, z)) \vee (...)$. From the first line, we can infer that after the a_b_r reaction the oxygen of the alcohol forms a single bond with the metal of the base (they correspond to the first and the fourth argument of the a_b_r action), and from the second line we derive that the oxygen of the alcohol molecule and its hydrogen cleave their bond (they correspond to the first and the second argument of the a_b_r action).
- the substrates and the products associated to X are used for individuating the chemical classes involved in the reaction, that will be represented as SC abbreviations, i.e, situation-dependent predicates that describe the molecular structure of interest. For example, the abbreviation for alcohol R−OH is: $Alcohol(o, h, s) \stackrel{def}{=} Hydroxy(o, h, s) \wedge \exists c(Alkyl(c, s) \wedge Bond(o, c, s))$ where $Hydroxy(o, h, s)$ and $Alkyl(c, s)$ are defined with further abbreviations.

Thirdly, the algorithm analyzes the specific chemical case and leads to the definition of the initial theory D_{S_0} (reflecting the instantiation of the values for the *Bond* and *DoubleBond* fluents in in the initial situation S_0), and of a *goal condition* representing the target compound to achieve. We use again abbreviations for representing goal molecules of the chemical case. For example, the formula $Goal(s) \stackrel{def}{=} (\exists o, h) Alcohol(o, h, s) \wedge (\exists cl, c) Ethyl\_Chloride(cl, c)$ states that our goal is to reach a situation s in which there are atoms identifying an alcohol molecule and an ethyl chloride.

At the heart of the Generation Layer lies the *PROLOG Planner*. This software module reasons on available molecules and chemical reactions formalized in SC for identifying a chain of reactions whose execution leads to a target compound. The search algorithm relies on a simple iterative deepening depth-first planning algorithm with declarative heuristics built ad-hoc for reducing the search space and help the program to find the synthesis route more quickly. The declarative heuristics are domain specific, but not planning problem instance specific. These heuristics identify duplicate unnecessary actions, as well as the irrelevant actions to the actions that have been taken so far. With this knowledge at hand, the PROLOG planner can avoid useless actions, and thus reduce the search space, which ultimately results in expediting finding the correct sequence of actions satisfying the goal. The interested reader can refer to [16] for more information about the details of the SC formalization and of the declarative heuristics.

If the PROLOG planner fails to generate a chemical process (it may happen that no declarative heuristic has been still developed for the specific chemical case), or the generated process is of insufficient quality (it possibly includes more reactions than the

ones expected), the task of process design is delegated to one or more state-of-the-art-planners. To this end, we developed a *Planners Manager* component, that takes as input the PDDL specification of the chemical reactions and of the chemical case as sent from the *XML-to-PDDL* component and invokes some external planners plugged to it. The *XML-to-PDDL* module works similarly to the *XML-to-SC*'s one. Specifically, it produces two files conforming with PDDL 2.2. The first file contains the PDDL *Planning Domain*, where *(i)* each generic reaction and its associated information (i.e., substrates, products and catalysts) are represented with a PDDL action schema, *(ii)* chemical bonds (respectively, double bonds) are translated in basic predicates and *(ii)* chemical classes are defined as derived predicates. The second file is the *Planning Problem*; here, the initial state is obtained by instantiating the existing bonds between the starting molecules, and the goal condition is inferred by the target compound as specified in the chemical case. It is worth noting that the PDDL Planning Problem roughly corresponds to the initial theory D_{S_0} in BATs, the PDDL Planning Domain roughly corresponds to PAs and SSAs in BATs, and PDDL derived predicates correspond to SC abbreviations. The Planners Manager allows to interact with several planners that are able to accept as inputs a planning domain and a planning problem defined with PDDL 2.2.

Once a plan satisfying the goal condition has been obtained, it is converted in a BPMN process through a *Translator* component. A BPMN chemical process is a simple sequence of activities that reflects the chain of reactions required for transforming the starting molecules to the goal compound. Once obtained, we store the chemical process in the Library of Chemical Processes and we return it to the chemist. Our approach assumes that a chemist contributes to the system by specifying the chemical case and the reactions step-by-step. If no plan is found, this suggests there is some missing element in the set of starting molecules/reactions used for the chemical process generation. In such a case, an empty process (one with no activities) is returned to the chemist, and an error log (associated to the chemical case) is recorded into the library. At this point the chemist may refine the chemical case or the set of reactions to obtain better solutions.

5 Experiments

In order to investigate the feasibility of our planning-based approach, we performed some testing to learn the time needed for designing a chemical process with different configurations of the starting molecules. The experiments were performed on a machine with 2.30 GHz CPU and 12 GB RAM. We ran our tests using our PROLOG Planner and two state-of-the-art planners, specifically, Fast-Downward [9] and Roamer [13]. Fast-Downward [9] is a progression planner that uses hierarchical decompositions of planning tasks for computing its heuristic function, called the *causal graph heuristic*, which approximates goal distances by solving a hierarchy of "local" planning problems. The Roamer planner [13] builds on the Fast-Downward planner, and uses a best-first search in first iteration to find a plan and a weighted A* search to iteratively decreasing weights of plans. The experimental setup was run on variants of our case study. We represented 9 SC/PDDL actions (corresponding to 9 different chemical reactions), annotated with 2 relational predicates (for identifying existing bonds between atoms) and 36 abbreviations/derived predicates (for representing the different chemical classes possibly involved in the reactions). Then, we defined 10 different planning problems of varying complexity by manipulating the composition of the starting molecules in the chemical case. Specifically, in Table 1, each case refers to a starting stage that includes the following molecules: water H_2O, hydrochloric acid HCl, sodium hydride NaH, ethyl fluoride CH_3-CH_2-F and an *Ester* molecule, that changes from case to case. The experiments done on such starting stages are identifiable in the first 5 rows in Table 1. In order to increase the complexity of the planning problems, we added to the chemical cases duplicate water and HCl molecules (cf. the last 5 rows in Table 1). In all experiments, the goal is to reach a state in which there are atoms identifying an alcohol

Chemical Cases	Ester	Prolog Planner	Fast-Downward	Roamer
Case 1	Methyl Acetate	0.06	47.82	47.78
Case 2	Ethyl Acetate	0.12	69.44	69.28
Case 3	Isopropyl acetate	0.75	171.46	171.37
Case 4	Methyl Butyrate	0.13	39.01	38.89
Case 5	Butyl Butyrate	3.45	45.79	45.47
Case 1 + H_2O + HCl	Methyl Acetate	0.06	76.03	75.81
Case 2 + H_2O + HCl	Ethyl Acetate	0.13	111.23	111.01
Case 3 + H_2O + HCl	Isopropyl acetate	0.75	285.56	280.83
Case 4 + H_2O + HCl	Methyl Butyrate	0.13	58.29	58.09
Case 5 + H_2O + HCl	Butyl Butyrate	3.46	67.48	67.43

Table 1. Time performances (in seconds) of the PROLOG planner, Fast-Downward and Roamer.

molecule and ethyl chloride. A planner invoked with whatever of the presented cases generates a chemical process composed by a chain of 4 chemical reactions.

As can be understood from the data collected in Table 1, the performance of the PROLOG planner is significantly better than the other planners. This is in part due to the domain-dependent declarative heuristics that are used in the PROLOG planner. In general, constructing domain-dependent heuristics offers opportunities to tailor the mechanisms to the particular domain for far a greater efficiency. A second reason lays in the way the PROLOG planner searches for a path satisfying the goal condition; it reasons on-the-fly on the available knowledge while generating the plan through its declarative heuristics, avoiding to build any intermediate structure to be exploited during the planning task. On the contrary, the state-of-the-art planners we have tested rely on a *translator* that converts the planner input from PDDL into a multi-valued state representation and on a *grounding algorithm* used for instantiating operators and axioms (e.g., derived predicates) of the planning domain into a grounded transition system. Finally, a *search engine* exploits the transition system just built for finding a satisfying plan. The main bottleneck is in the grounding algorithm, specifically when a large number of derived predicates need to be instantiated in the transition system, resulting in an exponential blow up of the space required for describing the planning problem. This is even more apparent when a larger starting stage is experimented (i.e. when duplicate water and HCl exist), where the performance of the PROLOG planner is almost intact while the state-of-the-art planners experience additional performance deficiency. This can be explained by direct relation of size of the starting stage with the time needed for the state-of-the-art planners to build the transition system and employ the domain independent heuristics for finding the solution.

To explore how close are the state-of-the-art planners to their limits, we extended our set of planning instances with an additional reaction: combustion of methanol CH_3-OH. In this reaction, all atoms of two methanol molecules and three O_2 molecules change bonds. Therefore, the action representing this reaction needs many arguments (18, to be precise). The initial stage molecules include additional three O_2 molecules and the domain description was extended with two additional abbreviations/derived predicates defining methanol and oxygen molecules. We observed that neither Fast-Downward nor Roamer were able to solve any of the instances of this extended set of planning instances, as they ran out of memory in their compilation stage. This was somewhat expected because these planners rely on grounding that leads to combinatorial explosion when actions have many arguments.

6 Related Work

The AI community has been involved with research on process management for several decades. While BPM has concentrated on business and manufacturing processes, the AI community has been motivated primarily by domains that involve active control of computational entities and physical devices (e.g., robots, antennas, etc.). An interesting

report that describes how techniques from AI could be leveraged to provide several of the advanced process management capabilities envisioned by the BPM community is shown in [17]. Surprisingly, to the best of our knowledge, AI techniques have never been applied for the purpose of modeling chemical reactions and automatically designing chemical processes, except for our previous work [16] and for the *Pathways* domain, that was introduced in the International Planning Competition (IPC) in 2005.

However, Computer-assisted organic synthesis (CAOS) is not a new field. CAOS aims to use computers to help chemists in the process of designing multi-step synthesis of organic compounds. It has been an active area of research for a long time and there has been an ongoing progress in developing new methodologies or expanding existing methodologies in the area of CAOS. The first synthesis system was organic chemical simulation of synthesis (OCSS) introduced in [4] which is based on retrosynthetic analysis. Retrosynthetic analysis has close connections with the well-known notion of regression in SC [20]. Soon after, LHASA [5] was developed which was extension of OCSS by incorporating a more complex strategy system which included multi-step plans based on useful reactions. Prior to 80s, CAOS systems were based on retrosynthesis approach but in 80s other approaches emerged. SYNGEN [10] was published with the main goal of automatically generating the shortest synthetic route for a given target structure. Formal-Logical Approach is discussed in [21] that focuses on reaction design problems. [23] discusses SYMBEQ computer program created for the search of novel types of organic reactions and is based on the formal-logical approach. More recently, Route Designer is discussed in [11]. In Route Designer, rules describing retrosynthetic transformations are automatically generated from reaction databases, which ensure that the rules can be easily updated to reflect the latest reactions in the literature.

The distinctive feature of our approach consists of representing *generic chemical reactions*, as opposed to specific instances of chemical reactions, like happens in the Pathways domain and in many CAOS systems. In other words, one generic reaction in our approach subsumes many instances of specific reactions. Additionally, we represent internal mechanism of reactions, i.e., we reason at the level of changing bonds between atoms. This aspect adds discovery potential to our approach, allowing the possibility to search for new chemical compounds and new drugs.

7 Conclusion

In this paper, we presented a planning-based approach and a framework for automatically designing chemical processes. We adapted the SC and PDDL to the task of modeling molecules and capturing semantics of generic chemical reactions, and we employed a PROLOG planner and two state-of-the-art planners to solve some process design problems. A full implementation of the approach is currently on going.

Our approach brings forth notable advantages such as being able to represent generic reactions and to reason about the most elementary interactions between molecules. Another advantage relies in the declarative representation of the chemical domain in SC/PDDL, which makes it easily expandable and flexible. Our experiments, despite being preliminary, demonstrate that some state-of-the-art planners may be employed (with some limitation) to solve process design problems. At the same time, they provide a successful case study of applying AI methods to solve problems in the life sciences. From the BPM perspective, we take inspiration from existing research works dealing with the synthesis of business process models [1,8,14] and we devise an approach that is able to automatically generate new chemical processes on the basis of some existing contextual data (i.e., information about molecules and reactions), without any underlying process model pre-defined in advance. We underline that the proposed approach is not limited to chemical processes design; it can be customized and used to solve other computational problems as well. To this end, we think that it can be particularly useful for those knowledge-intensive scenarios where the definition

of a process model strictly depends on the actual contextual information, by making unrealistic its definition in advance. As for future work, we intend to scale up the experiments by significantly enlarging the library of generic chemical reactions, and try more combinatorial problem instances. We are also planning to formalize further chemical conditions (e.g., temperature, energy, etc.) that are usually required for executing reactions. Finally, we are studying how to augment our reasoning approach to design chemical processes that allow reactions to be executed in parallel.

Acknowledgements. The work of Andrea Marrella has been partly supported by the SAPIENZA grants SUPER and "Premio Ricercatori Under-40".

References

1. Barba, I., Del Valle, C., Weber, B., Jiménez, A.: Automatic generation of optimezed business process models from constraint-based specifications. IJCIS (2013)
2. BPMI.org and OMG: Business Process Modeling Notation - Final Specification Ver.2.0. http://www.omg.org/spec/BPMN/2.0/PDF/ (2011)
3. Cook, A., Johnson, A.P., Law, J., Mirzazadeh, M., Ravitz, O., Simon, A.: Computer-aided synthesis design: 40 years on. Wiley 2(1), 79–107 (2012)
4. Corey, E.J., Wipke, W.T.: Computer-assisted design of complex organic syntheses. American Association for the Advancement of Science 166(3902), 178–192 (1969)
5. Corey, E.J., Wipke, W.T., Cramer, R.D., Howe, W.J.: Computer-assisted synthetic analysis. facile man-machine communication of chemical structure by interactive computer graphics. J. of the American Chem. Society 94(2), 421–430 (1972)
6. Di Ciccio, C., Marrella, A., Russo, A.: Knowledge-intensive Processes: An Overview of Contemporary Approaches. In: KiBP (2012)
7. Edelkamp, S., Hoffmann, J.: PDDL2.2: The Language for the Classical Part of the 4th International Planning Competition. Tech. rep., Albert-Ludwigs-Universitt Freiburg (2004)
8. Ferreira, H.M., Ferreira, D.R.: An integrated life cycle for workflow management based on learning and planning. International Journal of Cooperative Information Systems 15 (2006)
9. Helmert, M.: The fast downward planning system. J. AI. Res.(JAIR) 26, 191–246 (2006)
10. Hendrickson, J.B., Grier, D.L., Toczko, A.G.: A logic-based program for synthesis design. J. of the American Chem. Society 107(18), 5228–5238 (1985)
11. Law, J., Zsoldos, Z., Simon, A., Reid, D., Liu, Y., Khew, S.Y., Johnson, A.P., Major, S., Wade, R.A., Ando, H.Y.: Route designer: A retrosynthetic analysis tool utilizing automated retrosynthetic rule generation. J. of Chem. Inf. and Mod. 49(3), 593–602 (2009)
12. Leymann, F., Roller, D.: Production workflow: concepts and techniques. Prentice Hall (2000)
13. Lu, Q., Xu, Y., Huang, R., Chen, Y.: The roamer planner random-walk assisted best-first search. García-Olaya et al.(2011) pp. 73–76 (2011)
14. Marrella, A., Lespérance, Y.: Synthesizing a library of process templates through partial-order planning algorithms. In: BPMDS'13 (2013)
15. Marrella, A., Mecella, M.: Continuous planning for solving business process adaptivity. In: BPMDS'11 (2011)
16. Masoumi, A., Soutchanski, M.: Reasoning about chemical reactions using the situation calculus. In: 2012 AAAI Fall Symposium Series (2012), http://www.aaai.org/ocs/index.php/FSS/FSS12/paper/view/5635/5824
17. Myers, K., Berry, P.: Workflow Management Systems: An AI Perspective. AIC rep. (1998)
18. Nau, D., Ghallab, M., Traverso, P.: Automated Planning: Theory & Practice. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA (2004)
19. Reichert, M., Weber, B.: Enabling Flexibility in Process-Aware Information Systems - Challenges, Methods, Technologies. Springer (2012)
20. Reiter, R.: Knowledge in Action. Logical Foundations for Specifying and Implementing Dynamical Systems. MIT (2001)
21. Tratch, S.S., Zefirov, N.S.: Systematic search for new types of chemical interconversions: Mathematical models and some applications. J. of Chem. Inf. and Comp. Sc. 38(3) (1998)
22. Weske, M.: Business Process Management: Concepts, Languages, Architectures. Springer-Verlag New York, Inc., Secaucus, NJ, USA (2007)
23. Zefirov, N.S., Baskin, I.I., Palyulin, V.A.: Symbeq program and its application in computer-assisted reaction design. J. of Chem. Inf. and Comp. Sc. 34(4), 994–999 (1994)