Title

ChemStor: Using Formal Methods To Guarantee Safe Storage and Disposal of Chemicals

Journal

Journal of Chemical Information and Modeling, 60(7)

ISSN

1549-9596

Authors

Ott, Jason
Tan, Daniel
Loveless, Tyson
et al.

Publication Date

2020-07-27

DOI

10.1021/acs.jcim.9b00951

Peer reviewed

ChemStor: Using Formal Methods to Guarantee Safe Storage and Disposal of Chemicals

Jason Ott,[†] Daniel Tan,[†] Tyson Loveless,[†] William H. Grover,[*,‡] and Philip Brisk[*,†]

[†]*Department of Computer Science and Engineering, Winston Chung Hall, Room 351, University of California, Riverside, Riverside, CA, 92521 USA*

[‡]*Department of Bioengineering, 219 Materials Science and Engineering Building, University of California, Riverside, Riverside, CA 92521 USA*

E-mail: wgrover@engr.ucr.edu; philip@cs.ucr.edu

Abstract

While safe chemical storage and disposal are simple in principle—users should read safety specifications and place chemicals in appropriate cabinets or collection points—high-profile incidents involving improper storage and disposal of chemicals continue to occur. This paper introduces *ChemStor*, an open-source, automated computational system that can guarantee (mathematically verify a system is correct with respect to its specification), with regard to prescribed constraints, safe storage and disposal of chemicals used in academic, industrial, and domestic settings. *ChemStor* borrows concepts from formal methods—a branch of computer science capable of mathematically proving a specification or software is correct—to safely store or dispose of chemicals. If two or more chemicals can be combined in the same cabinet without forming possibly dangerous combinations of chemicals (while observing cabinet/shelf space constraints), then *ChemStor* determines that the storage configuration is safe. Likewise, if chemicals can be added to an existing disposal container without forming possibly dangerous combinations of chemicals (or exceeding the volume of the container), then *ChemStor* determines that the disposal configuration is safe. *ChemStor* accomplishes this by first building a chemical interaction graph, a graph that describes which chemicals may interact with each other based on their Reactivity Groups as determined by the United States Environmental Protection Agency. Next, *ChemStor* computes the chromatic number of the graph, the smallest number of colors used to color the graph such that no two vertices (chemicals) that share an edge (an interaction) share the same color. *ChemStor* then assigns all the chemicals of each color to a storage or disposal container after confirming that there is enough space in the container. These steps are encoded into a series of Satisfiability Modulo Theory equations, and *ChemStor* uses an industry-standard tool to try to find a valid solution to these equations. The result is either a solution which dictates exactly where to store or dispose of each chemical, or an indication that no safe storage or disposal configuration could be found. To demonstrate the feasibility of *ChemStor*, we used the tool to analyze ten real-world chemical

storage and disposal incidents that led to injuries or destruction of property. In each case, *ChemStor* quickly and successfully identified a proper chemical disposal or storage configuration that would have prevented the incident. In the future, *ChemStor* may be integrated with electronic laboratory notebooks, voice assistants, and other emerging technology to protect users of chemicals in labs, workplaces, and homes.

1 Introduction

Many common chemicals can undergo dangerous reactions when combined with incompatible chemicals during storage or disposal. For example, millions of tons of nitric acid are produced every year, making it a ubiquitous chemical in many research and industrial settings. Likewise, millions of tons of organic solvents are produced every year and used in many different applications. But when nitric acid is mixed with organic solvents, hazardous chemical products, fires, and explosions can result. While all chemists are (hopefully!) trained to avoid *intentionally* mixing nitric acid and organic solvents, accidents sometimes occur when these chemicals are *unintentionally* mixed in a chemical storage location or a waste disposal container.[1–4] Furthermore, nitric acid and organic solvents are just a few of the millions of chemicals that can undergo dangerous reactions when combined during storage or disposal. Incidents due to improper storage or disposal of chemicals occur with alarming frequency,[5–9] with consequences like second-degree burns[1] and destroyed laboratories.[6]

Efforts to improve chemical safety generally fall into two categories—*system-based approaches* and *behavioral approaches*—both of which have limitations in avoiding storage- and disposal-related incidents:

- **System-based solutions** focus on building systems (often computer-based) that aid research labs in managing many different administrative functions involving chemicals. These functions may include generating various state and federal compliance reports, automatically issuing a purchase order should inventory of a chemical fall below preset

levels, and sharing inventory among collaborating laboratories.[10–14] Some Chemical Inventory Management Systems (CIMS) employ simple safety features, like the ability to parse a chemical's Materials Safety Data Sheet (MSDS) to inform the researcher how to properly store the chemical. *Chempliance*[14] offers some guidelines on chemical disposal practices but is lacking any guarantees with respect to safety and does not track the volumes of chemicals.

- **Behavioral systems** focus on training and the human aspects of safety. Some of these systems train employees to focus on "safety first,"[15,16] while others aim to identify disparities between institutional and individual beliefs about safety.[17,18] These approaches focus on systemic issues that, while important, are sometimes relatively abstract and far removed from the specific day-to-day decisions faced by a researcher, like where to store a particular new chemical, or where to dispose of a certain waste chemical.

In summary, existing methods for improving chemical safety can inform researchers about general best practices, but fail to provide real-time guidance on specific storage and disposal decisions.

This paper introduces *ChemStor*, an open-source, automated chemical storage and disposal system that is provably safe with respect to proper laboratory safety protocols. For a given set of chemicals and containers, *ChemStor* either provides a specific storage or disposal configuration that is safe, or informs the user that no safe storage/disposal configuration is possible. Specifically, *ChemStor* informs users which specific cabinet or bin should be used to store or dispose of each chemical, thereby easing the burden of safety protocols that users must keep in their minds, while simultaneously minimizing the space required for chemical storage and disposal. *ChemStor* can be integrated with electronic laboratory notebooks, voice assistant tools, and many other existing and emerging technologies. Finally, *ChemStor* can enhance safety in a wide range of settings, not only research laboratories and industrial facilities, but also homes (where each year, *mixing incompatible pool cleaning chemicals* leads to an estimated 4,500 injuries alone[19]).

2 Overview of *ChemStor*

In this section, we summarize the operation of *ChemStor* in the context of a specific safety incident. In 1997, while one of the authors (WHG) was a student at the University of Tennessee, Knoxville, students in an undergraduate chemistry laboratory performed the Belousov-Zhabotinsky (BZ) reaction:

$$3\,CH_2(CO_2H)_2 + 4\,BrO_3^- \xrightarrow{Ce^{2+}} 4\,Br^- + 9\,CO_2 + 6\,H_2O \tag{1}$$

The BZ reaction is commonly studied in laboratory classes due to its unusual oscillatory nature. As is typically done, the students at UT Knoxville combined aqueous solutions of malonic acid and potassium bromate along with a cerium ammonium nitrate catalyst. When the reactants are combined as aqueous solutions, the reaction is benign, and the laboratory class concluded without incident. However, during post-lab cleanup, spilled reactants *in dry form* were swept from around the laboratory balances and into a waste container. This container was subsequently placed beneath a leaky sink, which began adding drops of water to the mixture of dry reactants after the laboratory was empty. The resulting reaction was extremely exothermic and caused a fire that resulted in significant damage to the laboratory. [5]

How could *ChemStor* have prevented this incident? More specifically, at the end of the laboratory class, after the dry reagents used in the BZ reaction were combined into the same waste container, how could *ChemStor* warn the teaching assistants that they should avoid any situation that might add water to this container (like placing the container beneath a leaky sink)?

The first step involves defining which chemicals are currently in which storage containers. Fig. 1A depicts the chemical storage situation at the end of the class, with the three dry reagents used in the BZ reaction (malonic acid, potassium bromate, and cerium ammonium nitrate) combined in a single chemical storage container, and the teaching assistant contemplating placing this container beneath a sink where water might be added to it.

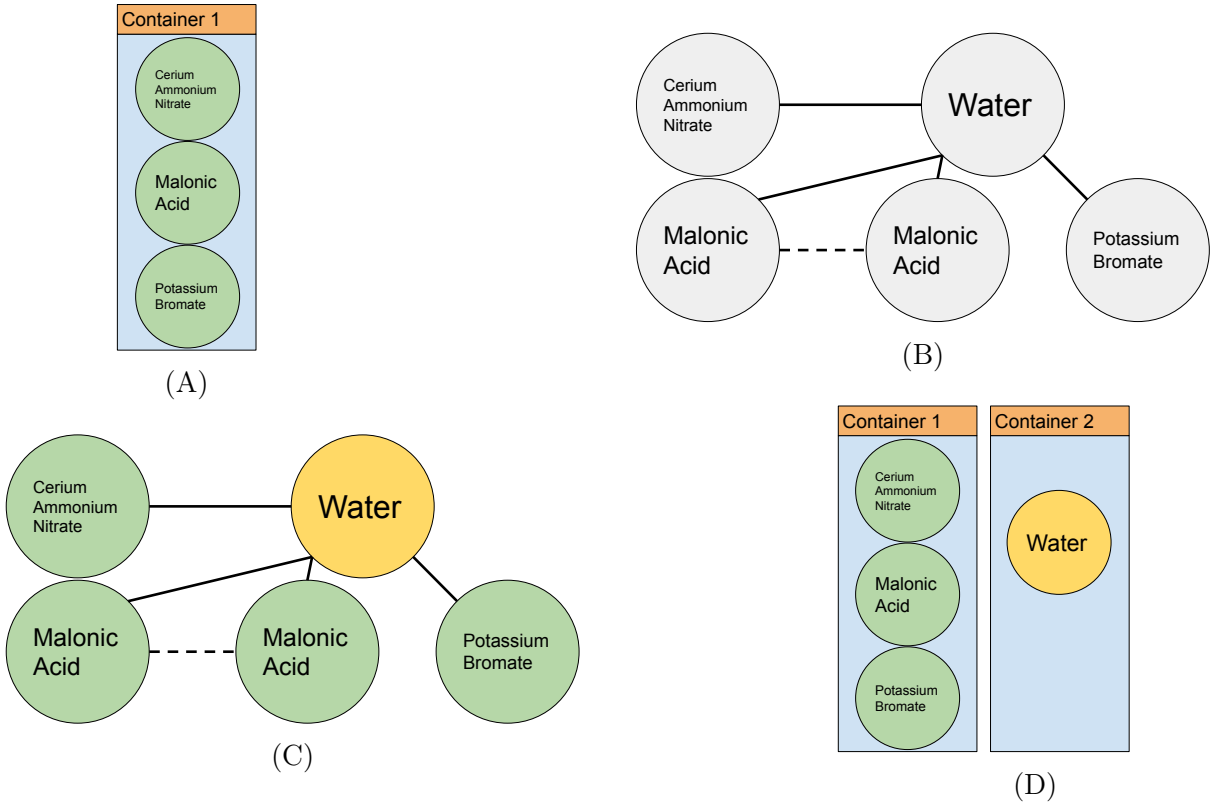


Figure 1: Using *ChemStor* to safely dispose of leftover reactants from performing the Belousov-Zhabotinsky (BZ) reaction. This simple scenario (based on real-life events that culminated in a lab-destroying fire[5]) begins with three reactants (cerium ammonium nitrate, malonic acid, and potassium bromate) combined in dry form in a single container **(A)**. A teaching assistant considers placing this container beneath a leaky sink drain, which will add water to the mixture. At this point, *ChemStor* constructs a chemical interaction graph **(B)** containing vertices for each chemical in the proposed mixture. In this graph, chemicals that may react with each other are linked with solid lines, and chemicals that are identical and can be combined are linked with dotted lines. After *ChemStor* calculates the chromatic number of the graph and colors the graph **(C)**, the chemicals can be safely added to different containers based on their vertex colors **(D)**. At this point, *ChemStor* would notify the teaching assistant that water should *not* be added to the container with the BZ reactants, the teaching assistant would avoid placing the container in a wet location, and a significant laboratory accident would have been avoided.

Next, *ChemStor* builds a *chemical interaction graph* G (Fig. 1B) containing 3 sets: a set of vertices, V; and two different sets of edges, E and A:

- The vertices, V, are the individual chemicals in this scenario. Specifically, we can define the vertices v = cerium ammonium nitrate, u = malonic acid, w = potassium bromate, and z = water. We can furthermore say that $v \in V$, $u \in V$, $w \in V$, and $z \in V$, or that v, u, w, z are members of, or are "in," the set of vertices V. (The "$\in$" symbol, as well as the other standard Boolean logic and set theory symbols used in this work, are defined in Table 1.)

- The set E is the set of edges that represent unsafe combinations of chemicals, or "interference" edges. Because mixing v, u, and w without z will not result in a dangerous reaction, it can be said that $(v, u) \notin E$; or that there is no edge between v and u. Similarly, there is no edge between the other combinations of the dry BZ reactants, so $(u, w) \notin E$ and $(v, w) \notin E$. Interference edges are denoted as solid lines between vertices in Fig. 1B. In this example the teaching assistant is considering possibly adding water to a container that already contains the dry BZ reagents (z and $z \in V$). Because z (water) *will* lead to a dangerous reaction when added to the combined dry BZ reagents, there are edges between each of the BZ reactants and water, so $(v, z) \in E$, $(u, z) \in E$, and $(w, z) \in E$, as shown in Fig. 1B.

- The set A is the set of edges that represent instances of the same chemical, properly defined as "affine" edges. The dotted line in Fig. 1B represents an affine edge. In this scenario, the edge represents some additional malonic acid (a) that requires disposal. Thus, $a \in V$, and $(u, a) \in A$ because both u and a are malonic acid and are chemically identical. The set of affine edges enables *ChemStor* to save chemical storage volume space by combining u and v, assuming the container for u or v has enough space. *ChemStor* represents this as $\forall (u, v) \in A$ ($\forall$ reads "for all").

Now that the chemical interaction graph G has been built, *ChemStor* can calculate the

Table 1: Common notation in set theory and Boolean logic.

Notation	**Plain English**	**Example**	**Outcome**
$\|A\|$	Cardinality	$\|A\|$	Number of elements in the set A
$\in$	Set membership	$1 \in \mathbb{N}$	Statement of fact — 1 is a member of the set of natural numbers
$\subseteq$	Subset or equal	$A \subseteq B$	Determines whether all the elements in set A are also in set B
$A \backslash B$	Set difference	$A \backslash B$	Returns the elements in B that are not in A
$\wedge$	Boolean AND	$X \wedge Y$	Evaluates true if and only if both predicates X and Y are true
$\vee$	Boolean OR	$X \vee Y$	Evaluates true if either/and both predicates X, Y are true
$\neg$	Boolean NOT	$\neg X$	Negates the predicate X. If X is true, then $\neg X$ evaluates to false
$\forall$	For all	$\forall a(a > 1)$	Definition of the natural numbers
$\exists$	There exists	$\forall(m \in \mathbb{N})\exists(n \in \mathbb{N})(n > m)$	The set of natural numbers continues *ad infinitum*

smallest number of containers required for safe disposal of the chemicals in G. *ChemStor* accomplishes this by calculating the graph's chromatic number, $\chi(G)$, a step in "coloring" the graph. Graph coloring is a mathematical problem that aims to color a graph's vertices with the minimal number of different colors, while guaranteeing that no two vertices that share an edge also share the same color. For safe disposal or storage of chemicals, the only edges that are of consequence are the interference edges, or the solid edges in Fig. 1B. This graph is trivially small and can be colored by hand using two colors, as shown in Fig. 1C. However, graph coloring belongs to a class of problems that are incredibly difficult to solve efficiently, the so-called *NP-complete* problems, so as more chemicals are added to the graph, the computational effort required to color the graph grows astronomically. The computational effort required to solve the graph coloring problem is proportional to 2^n, where n is the number of colors in the graph.

Finally, the graph's chromatic number (the number of different colors on the colored graph) denotes the minimum number of containers required to safely dispose of the chemicals in the graph. In this example, *ChemStor* recommends placing water in a separate container, not in the container with the dry BZ reagents (Fig. 1D). If this information was then communicated to the teaching assistant, they might avoid placing the container in a location where water could be added, thereby avoiding a significant laboratory accident.

3 Methods

In this section, we present details on the necessary data structures, algorithms, and constraints used by *ChemStor* to determine the safe disposal and storage of chemicals.

3.1 Chemical Compatibility

To obtain information about which types of chemicals are compatible or incompatible with each other, *ChemStor* relies on a chemical classification system created jointly by the US Environmental Protection Agency (EPA) and National Oceanic and Atmospheric Administration (NOAA).[20] This system categorizes over 9,800 chemicals into 68 reactivity groups that have similar properties. Mixing materials from certain reactivity groups can produce materials from other reactivity groups; for example mixing *acids* and *bases* induces a strong reaction that produces *salt* and *water*. The EPA/NOAA categorization assigns one of three outcomes to the combination of chemicals: *Incompatible*, *Compatible*, or *Caution*. We write $\mathsf{interact}(x, y) = \lightning$ if chemicals x and y cause an adverse reaction ($\lightning$) when mixed, or are *Incompatible*. Chemical combinations that result in *Caution* are either deferred to the expert user or treated as *Incompatible*. Finally, if either x or y (or both) are not classified with a reactivity group (in other words, they do not appear in the EPA/NOAA classification list), then the chemicals are marked as *Unknown* and *ChemStor* notifies the user that any possible storage or disposal solution would require review by an expert.

3.2 Chemical Interaction Graph

Let $G(V, E, A)$ be the *chemical interference graph* representing a storage problem. V is the set of chemicals we desire to store, E is the set of *interference edges* between incompatible chemicals, and A is the set of *affinity edges* between unique instances of identical chemicals. An interference edge (v, u) is added to E if, for any two chemicals v, u, $\mathsf{interact}(v, u) = \lightning$ as noted in § 3.1. An affinity edge (v, w) is added to A if v and w represent distinct instances of the exact same chemical (i.e., there is more than one container of a certain chemical we wish to store). The chemical interaction graph may be extended to include a set $P \subseteq V$ of vertices and $Q \subseteq E$ of edges representing chemicals already in storage and their corresponding edges, respectively.

3.3 Chemical Storage

Let $C = \{c_1, c_2, \ldots, c_k\}$ be the set of *cabinets* for chemical storage. Each cabinet contains a finite set of *shelves*. Let $S(c_m)$ denote the set of shelves within cabinet $c_m \in C$, where s_n^m denotes the n^{th} shelf in c_m. Each shelf $s_n^m \in S(c_m)$ has an immutable capacity, denoted $\mathsf{maxCapacity}(s_n^m)$ with which it can use to store chemicals. The capacity of each cabinet is the sum of the capacities of its shelves. The capacity of a shelf currently occupied by chemicals is $\mathsf{currCapacity}(s_n^m)$.

Affinity-adjacent chemicals may be combined into the same container. The cost of a container is orders of magnitude less than the cost of a cabinet; as such, we assume an infinite supply of containers but a finite supply of cabinets.

Let $\mathsf{currVolume}(x)$ denote the current volume chemical x occupies within its container, and let $\mathsf{combine}(y, z)$ be the volume of the combined quantities of chemicals y and z, which in either case may be less than the volume of their respective containers. Two instances of the same chemical v and w can be combined by coalescing their respective vertices in G.

3.4 Chemical Disposal

Let $D = \{d_1, d_2, \ldots, d_k\}$ be the set of *containers* for chemical disposal. Each container has a maximum volume associated with it, denoted $\mathsf{maxVolume}(d_m)$. The current volume of the container d_m is $\mathsf{currVolume}(d_m)$.

Thus, if $(v, u) \in E$, then v and u cannot be combined in the same container d_m (e.g., $\mathsf{interact}(v, u) = \lightning$), so a new container must be added: d_{m+1}. If $(v, u) \notin E \wedge \mathsf{vol}(v) + \mathsf{currVolume}(d_m) \leq \mathsf{maxVolume}(d_m)$, then v can be combined with u in the container d_m. Affinity-adjacent chemicals are assumed to be combined into the same container, assuming the maximum volume of the container allows it.

3.5 Characterization of a Solution to the Chemical Storage Problem

Given a chemical interference graph G, *ChemStor* computes a pair of functions $f : V \rightarrow \{1, 2, \ldots |C|\}$ and $g : V \rightarrow \mathbb{N}^+$ which assigns each chemical (vertex) to a specific storage location within a cabinet and on a shelf, or (cabinet, shelf). If $(f(v), g(v)) = (m, n)$, then chemical v is assigned to shelf s_n^m in cabinet c_m, $1 \leq n \leq |S(c_m)|$.

A legal chemical storage solution must satisfy the *Chemical Reactivity Constraint*, which states that two interfering chemicals cannot be stored in the same cabinet:

$$f(v) \neq f(u) \; \forall (v, u) \in E \tag{2}$$

A more permissive variant of the Chemical Reactivity Constraint allows two interfering chemicals to be stored in the same cabinet, but on different shelves:

$$\begin{aligned} p_1 &= f(v) \neq f(u) \\ p_2 &= f(v) = f(u) \wedge g(v) \neq g(u) \\ p_{12} &= p_1 \vee p_2 \; \forall (v, u) \in E \end{aligned} \tag{3}$$

A legal chemical storage solution must also satisfy the *Storage Capacity Constraint*, which states that the sum of the capacities of the containers assigned to each shelf in each cabinet cannot exceed that shelf's storage capacity:

$$\sum_{v \in V | f(v)=m,\ g(v)=n} \mathsf{currVolume}(v) \leq \mathsf{maxCapacity}(s_n^m)$$
$$1 \leq m \leq |C| \tag{4}$$
$$1 \leq n \leq |S(c_m)|$$

3.6 Satisfiability Modulo Theories

ChemStor uses a class of logical formula solvers, Satisfiability Modulo Theories (SMT), to solve a given storage or disposal problem instance. An SMT solver determines whether a problem instance is "decidable", or can be answered by a simple "true" or "false". SMT problems support linear inequalities (e.g., $x+5y-2z \leq 5$), equalities involving uninterpreted terms or functions (e.g., $f(u,v) = f(g(v),u)$), Boolean logic (e.g., $a \wedge b$), and in some cases quantifiers (e.g., $\forall a(a \in \mathbb{N})(a > 0)$).

SMT-based problems are expressed as a series of mathematical constraints. These constraints define the valid range of values variables can take for a solution. SMT equations are very expressive and can take one or many of the form(s) noted above. Once these equations are defined, they are used as input to an SMT solver. If the solver can find a solution which satisfies the constraints, it provides a model, or the values of all the variables. If no solution can be found, the solver simply returns "false".

3.7 SMT Constraints

To use a SMT solver to solve a *ChemStor* problem instance, we first convert the Chemical Storage Problem, described in § 3.5, into a set of SMT equations. Each chemical v must be assigned to exactly one shelf $s_{n_v}^{m_v}$ in exactly one cabinet c_{m_v}. We accomplish this using the

following constraints:

$$m_v \in \mathbb{Z}, 1 \leq m_v \leq |C| \tag{5}$$

$$n_v \in \mathbb{Z}, 1 \leq n_v \leq |S(c_k)| \wedge k == m_v \tag{6}$$

If v is a previously stored chemical, then the values for m_v and n_v are known *a priori* and are encoded as SMT constants.

The second constraint, the *Chemical Reactivity Constraint*, guarantees that no pair of chemicals stored in the same cabinet can interact dangerously, and can be expressed as an SMT constraint as follows:

$$\forall u, v \in V | m_u == m_v \; \mathsf{interact}(u, v) \neq \lightning . \tag{7}$$

The more permissive variant of this constraint, Eq. (3), guarantees that no pair of chemicals stored in the same shelf in the same cabinet can interact dangerously, and can be expressed as an SMT constraint as follows:

$$\forall u, v \in V | m_u == m_v \wedge n_u == n_v, \; \mathsf{interact}(u, v) \neq \lightning . \tag{8}$$

Finally, the *Storage Capacity Constraint* (Eq. (4)) expresses the Storage Capacity Constraint in a form that is already SMT-compatible.

3.8 Coalescing Strategy

As defined in § 3.2, an affinity edge $(u, v) \in A$ represents two containers that store identical chemicals. Let $t(u)$ denote the chemical "type" of u. In *ChemStor*'s case, the reactivity groups described in § 3.1 comprise the different "types" to which chemicals may belong, like

"acid" and "base." In Eq. (4), $\mathsf{maxVolume}(v)$ represents the volume of the container that holds chemical v. Let $\mathsf{currVolume}(v) \leq \mathsf{maxVolume}(v)$ denote the volume of the chemical held in the container. To reduce demands on limited storage space, it may be possible to consolidate multiple instances of the same chemical (u and v) into u's container if

$$\mathsf{currVolume}(u) + \mathsf{currVolume}(v) \leq \mathsf{maxVolume}(u). \tag{9}$$

Here, the user no longer needs to store v's container, and *ChemStor* can eliminate all of v's associated SMT constraints.

We implement this feature as a coalescing (vertex merging) operation applied to the chemical interference graph prior to calling the SMT solver; in practice, coalescing opportunities could be incorporated directly into the SMT formulation as well.

Fig. 2 illustrates coalescing. Here u represents 150 mL of hydrochloric acid in a 300 mL container and v represents 150 mL of hydrochloric acid in a 300 mL container. Without coalescing, the shelves would use a combined 600 mL of volume to store u and v. However, with coalescing, *ChemStor* combines them into a single container, reducing the storage requirement to 300 mL.

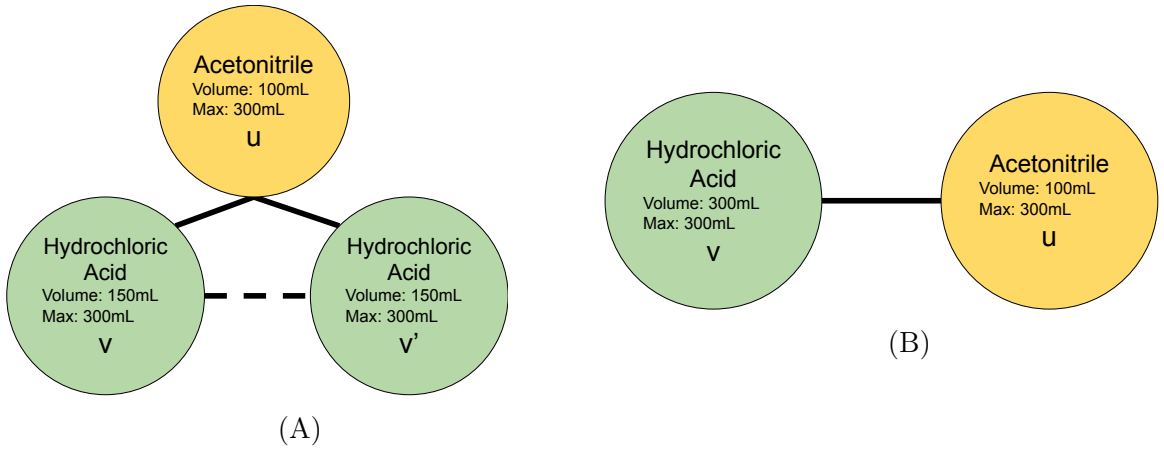


Figure 2: Demonstrating the coalescing strategy. The affine edge (dotted line) in **(A)** allows *ChemStor* to combine those chemicals into one vertex as shown in **(B)**, as long as the volumes $v + v' \leq \mathsf{maxVolume}(v, v')$.

3.9 De-Coalescing Strategy

In some cases, it may be necessary to *split* one chemical container into two or more containers; *ChemStor* addresses this behavior through *de-coalescing*. As a motivating example, suppose that a user tries to store 300 mL of hydrocholoric acid in a cabinet with three shelves, as shown in Fig. 3. Due to pre-existing chemicals allocated to storage, only 100 mL of space is available on each shelf. In this case, it makes sense to *split* the hydrochloric acid into three 100 mL containers; otherwise, a legal storage solution cannot be found. We implement this strategy using de-coalescing (vertex splitting).

ChemStor is allowed to de-coalesce a vertex v into a given number of p parts, each having a volume no more than $\mathsf{vol}(v)/p$. This is expressed using conditional notation within our solver; due to the inherent complexity of this constraint, we describe it conceptually here. If there are q shelves that do not interact negatively with v whose combined available capacity is greater than $\mathsf{vol}(v)$, but whose individual available capacities are smaller than $\mathsf{vol}(v)$, then we can split v into an equal number of p parts, where p is $\mathsf{vol}(v)$ divided by the greatest common divisor of the available capacities of the available capacities of the q shelves. We can then de-coalesce as described above, and the resulting p instances of chemical v can be stored on the q shelves either directly or after further coalescing.

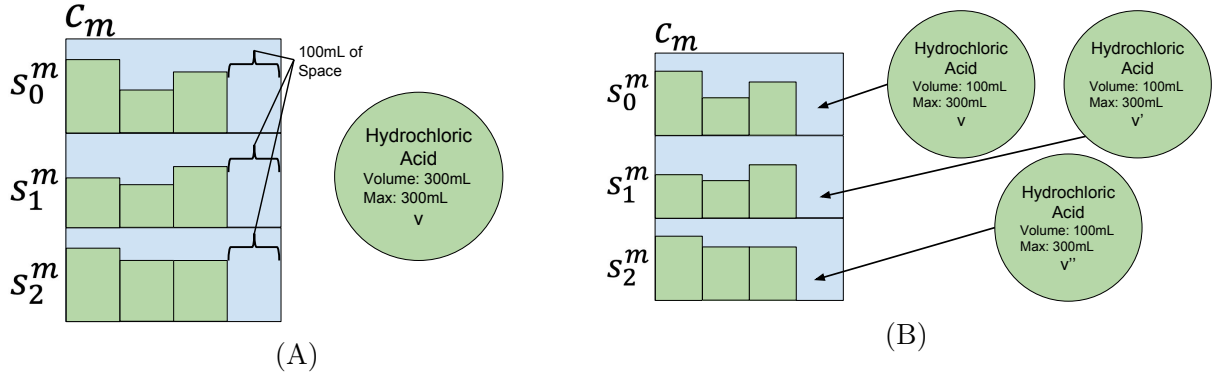


Figure 3: Demonstrating the de-coalescing strategy. To store a chemical whose volume exceeds the capacity of any one shelf **(A)**, *ChemStor* de-coalesces vertices and splits the chemical into p parts to derive a feasible storage configuration **(B)**.

3.10 No Solutions

There are instances when *ChemStor* might not converge to a legal solution (in this context, a "legal solution" is one whose constraints have been met, *e.g.*, all chemicals stored in a cabinet have no adverse reaction should they interact). In some cases, this may be in part due to constraints imposed by the set of chemicals pre-assigned to storage locations. One possibility is to unassign all of these vertices and generate a new SMT problem instance. If this second instance is successfully solved, then a legal storage solution has been found, albeit one that may require a significant rearrangement of chemicals stored in the cabinets. If the second instance cannot be solved, then the user is informed that no legal storage solution is possible using the existing resources.

4 Results

We implemented *ChemStor* using the Python programming language and z3[21] as our SMT solver. All experiments were performed on a 64-bit Windows 10 Dell Laptop with an Intel(R) Core(TM) i5-7200U CPU @ 2.50 GHz with 8.00 GB of RAM. The code is available at `https://github.com/lilott8/BioScript`.

To test the efficacy of *ChemStor*, we used it to reproduce a number of real-world destructive chemical storage and disposal incidents. In some cases, details regarding the storage resources were sparse or non-existent; in these examples, we made reasonable assumptions about common storage cabinet sizes and quantities, chemical volumes, and chemical taxonomies. For each real-world incident in Table 2, we averaged run times across 100 runs. In each case, *ChemStor* is able to derive a safe and valid storage or disposal solution in a fraction of a second.

Table 3 reports the ability of *ChemStor* to handle storage problems which necessitate the use of our coalescing and de-coalescing strategies (i.e., valid solutions are unable to be found without combining or splitting containers of chemicals). *ChemStor* was able to find a

Table 2: Results from using *ChemStor* to solve chemical storage and disposal problems from real-life incidents. In these incidents, faulty storage or disposal configurations caused lab fires, explosions, or human harm and incurred significant damages to lab spaces. All run times were averaged across 100 tests. *ChemStor* was able to find a safe chemical storage or disposal configuration for each incident in a few milliseconds.

Incident	Avg. Run Time(ms)	No. of Chemicals	No. of Cabinets	No. of Shelves	Solution Found
Tetracholorethylene+Nitric Acid[22]	5.523	2	1	4	Yes
Hexane Explosion[9]	2.105	7	1	7	Yes
Methanol+Nitric Acid[22]	2.386	2	1	2	Yes
Benzene+Urea+Benzotrichloride[22]	4.450	3	1	3	Yes
Lithium Aluminum Hydride Fire[7]	7.043	3	1	3	Yes
H_2O_2+Sulfuric Acid + Acetone[22]	10.086	3	1	4	Yes
Formaldehyde+Benzene[22]	11.338	3	1	4	Yes
Univ. of Tennessee, Knoxville Fire[5]	23.177	4	1	4	Yes
Broken Beaker of Barium Oxide[8]	1.272	11	3	3	Yes
Lab Fire at Ohio State University[6]	1.241	9	2	3	Yes

safe storage configuration for all the synthetic storage problems where one exists, denoted by "Yes" in the *Valid Solution Found* column. In the case of the two synthetic failures, *ChemStor* was unable to find a *safe* storage configuration, as was expected. The two failing test cases demonstrate problem instances where a valid solution is impossible and the likelihood of an unsafe incident is significant. In these two cases, the coalescing and de-coalescing strategies would not prove helpful as the volume constraint on a shelf prevents a chemical from being safely stored. As in Table 2, we average run times over 100 runs, and note that every test returns in a few milliseconds.

5 Conclusions

ChemStor generates safe storage and disposal configurations that are provably safe, given the chemicals have assigned EPA/NOAA reactivity groups. In all of our test cases, ranging from real-world to synthetic, *ChemStor* converged in less than one second, which indicates that realistic storage/disposal problem instances yield SMT problem instances that can be

Table 3: Synthetic tests demonstrating the efficacy of *ChemStor*'s coalescing and decoalescing strategies. All run times were averaged across 100 tests. We crafted tests for the edges case: restrictive placement, relaxed placement, coalescing success or failure, and decoalescing success or failure. If a solution could not be found, the corresponding column is marked with a "No".

Test	Avg Run Time (ms)	No. of Chemicals	No. of Cabinets	No. of Shelves	Solution Exists	Solution Found
Full Cabinet	0.64	5	1	4	Yes	Yes
Compatible Chemicals	39.54	5	3	4	Yes	Yes
Combine Two Tests	142.96	8	4	4	Yes	Yes
Pass Coalescing	4.94	1	1	1	Yes	Yes
Pass Decoalescing	10.60	1	1	3	Yes	Yes
Fail Coalescing	3.14	1	1	1	No	No
Fail Decoalesce	4.77	1	1	2	No	No

solved rapidly. This is important, as it enables *ChemStor* to provide real-time advice to users before dangerous storage and disposal mistakes are made.

In its current form, *ChemStor* has some significant limitations. For example, as noted earlier, *ChemStor*'s problem instance is difficult to solve; and because of the difficulty of the graph coloring problem, parallelization is of little help. While *ChemStor* is capable of accepting a problem instance containing millions of chemicals, problems of that magnitude might not converge on a solution quickly. Additionally, *ChemStor* doesn't account for chemical properties like concentrations or temperatures; obviously these properties heavily influence the reactivity of the chemicals. Also, the notion of a "cabinet" in *ChemStor* is abstract, and differentiating between, say, a refrigerator and a room-temperature shelf is an important distinction when storing a chemical with a flash point near room temperature. Finally, the *ChemStor* "container" is an abstract volume and does not capture the real-world container dimensions that dictate shelf or cabinet capacity. Future versions of *ChemStor* should address these shortcomings, and since *ChemStor* is an open-source project, we welcome others to add additional capabilities to the software and use it in their own projects.

In the near future, *ChemStor* can be incorporated into the various technological assistants that are gaining popularity in workplaces and homes. For example, by including *ChemStor*

in an electronic laboratory notebook, the notebook software could automatically suggest chemical disposal strategies after each experiment. Cameras in augmented reality systems could actively scan the workplace and use *ChemStor* to identify unsafe chemical storage situations before accidents occur, and microphones could listen for employees' questions about storage and disposal. These scenarios are not that far-fetched—software developers are already working on voice-based assistants for chemists,[23] and integrating *ChemStor* into these tools seems relatively straightforward. Finally, including *ChemStor*'s recommendations in home voice assistants like Apple's Siri and Amazon's Alexa could significantly reduce the number of chemical-related injuries and accidents that occur in homes.

Acknowledgement

This work was supported by the National Science Foundation under grants 1910878, 1740052, 1545097, 1536026, and 1351115.

References

(1) AIHA, internet, Accessed on 2019-06-29; https://www.aiha.org/get-involved/VolunteerGroups/LabHSCommittee/Incident%20Pages/Lab-Safety-Chemical-Exposures-Incidents.aspx.

(2) Texas Tech University, E. o. H.; Safety, internet, Accessed on 2019-05-24; https://www.depts.ttu.edu/research/integrity/lessons-learned/February-2015.php.

(3) University of Illinois, D. o. R. S. internet, Accessed on 2019-05-28; https://www.drs.illinois.edu/News/Nitric-Acid-Alert.

(4) Washington University in St. Louis, E. H.; Safety, internet, Accessed on 2019-05-21; https://wustl.box.com/s/my0l7melrkpzgdfneu7dcscnvve773l4.

(5) LETTERS. *Chemical & Engineering News Archive* **1998**, *76*, 4.

(6) Schulz, W. G. internet, 2005; http://www.ehs.ucsb.edu/files/docs/ls/Ohio_fire.pdf.

(7) AIHA, internet, Accessed on 2018-03-14; https://www.aiha.org/get-involved/VolunteerGroups/LabHSCommittee/Pages/Lithium-Aluminum-Hydride-Fire.aspx.

(8) University of California, C. f. L. S. internet, Accessed on 2018-03-22; https://ucla.in/2BAL5Gq.

(9) AIHA, internet, Accessed on 2018-03-15; https://www.aiha.org/get-involved/VolunteerGroups/LabHSCommittee/Pages/Shelf-Collapse-Causes-Spill-and-Fire.aspx.

(10) Gibbs, L. ChemTracker consortium–The Higher Education Collaboration for Chemical Inventory Management and Regulatory Reporting. *J. Chem. Health Saf.* **2005**, *12*, 9–14.

(11) Cournoyer, M. E.; Maestas, M. M.; Porterfield, D. R.; Spink, P. Chemical Inventory Management: The Key to Controlling Hazardous Materials. *J. Chem. Health Saf.* **2005**, *12*, 15–20.

(12) Santos, J. E. R.; Alfonso, F. N. N.; Mendizabal Jr, F. C.; Dayrit, F. M. Developing a Chemical and Hazardous Waste Inventory System. *J. Chem. Health Saf.* **2011**, *18*, 15–18.

(13) Foster, B. L. The Chemical Inventory Management System in Academia. *J. Chem. Health Saf.* **2005**, *12*, 21–25.

(14) Rappaport, J.; Lichtman, J. Ongoing Development of a Chemical/Biological Inventory

and Safety Management Solution for Temple University. *J. Chem. Health Saf.* **2005**, *5*, 4–8.

(15) Khan, F. I.; Amyotte, P. R. How to Make Inherent Safety Practice a Reality. *Can. J. Chem. Eng.* **2003**, *81*, 2–16.

(16) Cournoyer, M. E.; Maestas, M. M. Addressing Safety Requirements Through Management Walkarounds. *J. Chem. Health Saf.* **2004**, *11*, 12–16.

(17) Silva, S.; Lima, M. L.; Baptista, C. OSCI: an Organisational and Safety Climate Inventory. *Safety science* **2004**, *42*, 205–220.

(18) Cournoyer, M. E. A Risk Determining Model for Hazardous Material Operations: Part II. Probabilistic Safety Assessment and Management. 2004; pp 1534–1540.

(19) Pool Chemical Injuries Lead to Over 4,500 Emergency Department Visits Each Year. internet, Accessed on 2019-04-22; `https://www.cdc.gov/media/releases/2019/p0515-pool-chemical-injuries.html`.

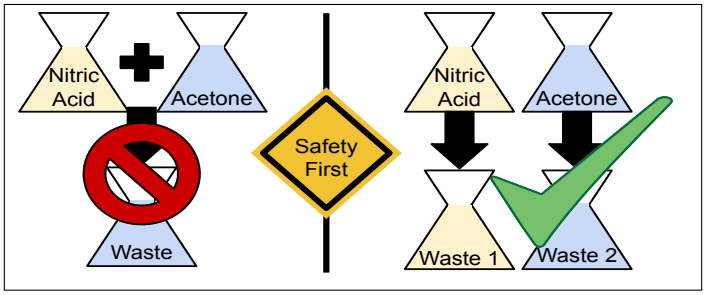
(20) Environmental Protection Agency & National Oceanic and Atmospheric Administration, internet, Accessed on 2016-09-12; `https://cameochemicals.noaa.gov/`.

(21) Bjørner, N.; de Moura, L. Z3: An Efficient SMT Solver. *Tools and Algorithms for the Construction and Analysis of Systems,(TACAS'08)* **2008**,

(22) Ott, J.; Loveless, T.; Curtis, C.; Lesani, M.; Brisk, P. BioScript: Programming Safe Chemistry on laboratories-on-a-Chip. *Proceedings of the ACM on Programming Languages* **2018**, *2*, 128.

(23) Mullin, R. Alexa and your Phone are Getting Schooled in Chemistry. *Chemical & Engineering News* **2019**, *97*.

Graphical TOC Entry



Contents

Some journals require a graphical entry for the Table of Contents. This should be laid out "print ready" so that the sizing of the text is correct.
Inside the `tocentry` environment, the font used is Helvetica 8 pt, as required by *Journal of the American Chemical Society*.
The surrounding frame is 9 cm by 3.5 cm, which is the maximum permitted for *Journal of the American Chemical Society* graphical table of content entries. The box will not resize if the content is too big: instead it will overflow the edge of the box.
This box and the associated title will always be printed on a separate page at the end of the document.