Container Trajectories

An Introduction via Case Factors in Trade Secrets Law

Dominik Rabiej

May 18, 2006

Advanced Undergraduate Project
6.AUP
Spring 2006

Abstract

This paper presents the concept of *container trajectories* by way of an illustrative example from the domain of trade secrets law. A container trajectory is an extension of Jackendoff's trajectory concept. It has the added benefits of being able to operate within informational environments, indicate state and add filtering capabilities to trajectories. Case factors are used by AI in law to indicate a particular salient characteristic of a case. By expressing case factors using the language of container trajectories, case factors can now interact with and depend on each other. AI thus gains a new representational vocabulary to describe law, and law gains the possibility that natural language processing might be able to produce a system capable of reading case briefs, forming container trajectories, deriving case factors and automatically providing a set of relevant cases to support either the plaintiff or defendant.

1. Introduction

1.1. Law and Artificial Intelligence: A Natural Synergy

Studying law from the perspective of artificial intelligence confers benefits to both fields. Artificial intelligence gains a well-practiced means of describing information while law gains new understanding of its own underlying structure, along with the possibility of insights into hitherto undiscovered patterns.

AI and Law is a rich source of research in topic areas such as negotiation, decision-making, e-commerce, natural language, information retrieval and extraction, and data mining [1]. In particular, this project focuses on applying principles of AI in the legal arena, for several aims. Extending the artificial intelligence concept of *trajectories* to be able to manipulate legal *case factors* gives AI more tools to reason with. It also serves a valuable purpose in setting a reachable target for natural language processing to strive for: law has the advantage of being a well-documented, systematic field of written work.

1.2. What are trajectories?

The concept of a trajectory, first introduced by Jackendoff [2], is simple to illustrate. Consider this English sentence:

The mouse went under the table.

This sentence is unclear. Does the mouse go under the table and stop there? Or does the mouse simply run under the table and keep going? A trajectory settles this ambiguity by applying one of five basic concepts of motion: through, into, out of, toward, and away. If the mouse went under the table and stopped there, that would be an *into* trajectory. If instead the mouse simply went under the table on its way elsewhere, that would be *through* trajectory.

These five basic concepts are the fundamental building blocks of trajectories. Jackendoff's work centered on describing the physical world with language, and thus there are certain implicit assumptions present in his definition of trajectories. Specifically, an important distinction is that in the physical world objects cannot be in two places at once. If the mouse is under the table it cannot be simultaneously on top of the very same table. This restriction will prove to be an important distinction between Jackendoff's trajectories and this research's container trajectories.

1.3. Case Factors

Case factors are "stereotypical patterns of facts that tend to strengthen or weaken a side's legal claim" [3]. In AI, they are essentially tags or flags applied to case by a human editor. They have no internal structure themselves and merely serve to highlight an aspect of the case that a human thought noteworthy. For example, if in a trade secrets legal case a bartender came up with a new recipe and then made sure it was always kept secret under lock and key, a human editor should assign the *Security-Measures* factor to indicate that in this case, there were security measures taken to protect the trade secret.

1.4. Container Trajectories: Factors and Trajectories Combined

How then can the concepts of trajectories and case factors be united? Furthermore, what is the benefit in uniting them at all? These are the questions that this research examines and addresses. By uniting trajectories and case factors via *container trajectories*, AI and law both gain a new language for dealing with legal information. Unlike trajectories, container trajectories work well in information-based environments.

Unlike case factors, container trajectories are flexible and modular, which means that a sufficiently advanced natural language processing system could automatically derive them from a legal case brief. Essentially, by formalizing information into container trajectories, factors become "naturally derivable" from trajectories. Factors cease to be arbitrary human-edited creations and become labels for common trajectory patterns. This leads to the exciting vision of a natural language processing system capable of reading case briefs, forming container trajectories, deriving case factors and automatically providing a set of relevant cases to support either the plaintiff or defendant.

2. Container Trajectories: An Illustrative Example

2.1. The Mason Case

> Tony Mason, the plaintiff, developed a cocktail he dubbed "Lynchburg Lemonade." Since Mason took some measures to protect his recipe's secrecy, and since his was the only tavern producing this drink, we say factors **Security-Measures** and **Unique-Product** apply; both tend to favor the plaintiff. On the other hand, Mason disclosed his recipe in negotiations with a sales agent of the defendant, Jack Daniel's Distillery, which started marketing an identical cocktail (thus the **Identical-Products** factor) without compensating Mason. Thus **Disclosure-In-Negotiations** applies, a factor that tends to favor the defendant. The agent was aware, however, that the recipe was a "secret formula," so **Knew-Info-Confidential**, also applies, tending to favor the plaintiff. Finally, the recipe could have been obtained by reverse engineering the cocktail; **Info-Reverse-Engineerable**, applies and favors the defendant. [3]

This case will serve as an example to illustrate container trajectories as applied to case factors in the context of trade secrets law.

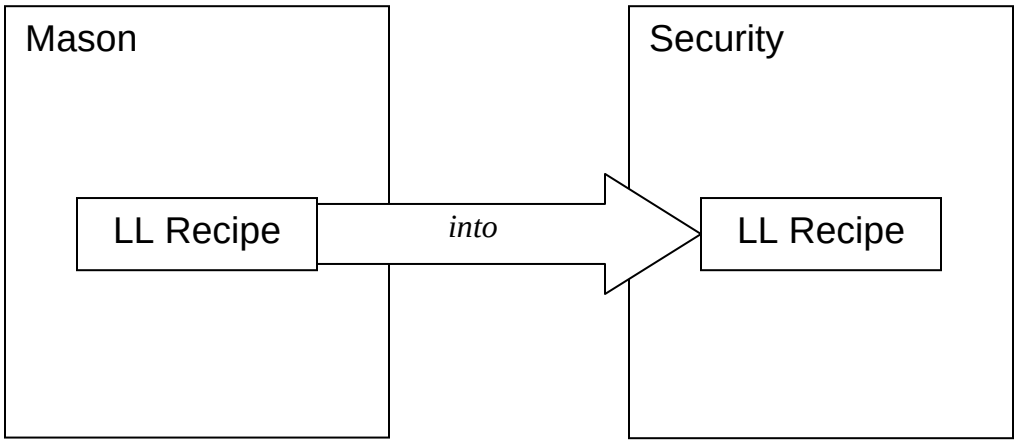
2.2. Defining a Container Trajectory

Before we embark on defining container trajectories, it is worth stepping back a bit and considering the nature of law itself. Law is effectively enshrined arguments: one side presents its views and what it sees as necessary conclusions and then the other side

does the same. A judge and jury weigh the concerns, evaluate the strength of the arguments on their merits and then reach a decision.

In 1958, Toulmin created a useful model for representing argumentation [4]. His model has six primary components: claim, data, warrant, backing, qualifier and rebuttal. Consider the Mason case above. Tony Mason's claim is that Jack Daniel's has stolen his trade secret; the data for this claim is that the Distillery is marketing the cocktail without compensating him. The warrant for the data is essentially the frame that makes the data valid for consideration. In this case, the warrant is that taking something from someone and not paying them for it qualifies as stealing. The backing is that which gives the warrant effectiveness; here the body of trade secrets law serves as the backing for the warrant that stealing is wrong. Finally, a qualifier restricts the impact of a claim while a rebuttal directly contradicts it by challenging its data, warrant or backing.

How then do container trajectories take into account for these principles of argument while representing case factors? Let's examine a container trajectory for the Security-Measures factor in the Mason case:



This is a very simple diagram representing the concept that Mason wanted to secure the information contained within Lynchburg Lemonade recipe.

There are actually three containers in this container trajectory. Mason is the first container, containing the LL Recipe. The second container is Security, which also

contains the LL Recipe. Finally, the LL Recipe itself is a third container; it is only one container, even though there are two boxes representing it in the diagram.

The diagram is called container trajectory because, in it, a container follows a trajectory that relates one container to another. The diagram shows the LL Recipe container, resident in the Mason container, proceeding along the *into* trajectory thus moving into the Security container. The five basic types of trajectories remain the same: through, into, out of, toward and away.

2.3. Information Containers

In dealing with trade secrets law, the primary concern is information. Thus when the LL Recipe in the diagram moves *into* the Security container, it does not necessarily move *out of* the Mason container. Merely placing the LL Recipe information into the Security container does not make Mason forget the LL Recipe information!

This conflicts squarely with the way Jackendoff's trajectories work: if an object moves into something, it must move out of something else. The reason for this conflict is because Jackendoff restricted his trajectories to only dealing with the physical world.

How then can we resolve this conflict?

First, we must realize that all trajectories but through have two components: an into trajectory has an explicit into component as well as an implicit out of component. Similarly, an away trajectory has an explicit away component and an implicit towards component. In the physical world, the implicit component is not always formally defined, but in an information-based world, we can create containers to serve as implicit components.

Trajectories interact with two containers. One is the primary reference of the trajectory and the other is the secondary reference. The two components of a trajectory correspond to the two containers it interacts with: the explicit trajectory component interacts with the primary reference container and the implicit trajectory component interacts with the secondary reference container.

In our *LL Recipe into Security* example, the explicit trajectory component is into, the implicit is out of. The primary reference is Security, the secondary reference is

Mason. Thus the into trajectory component is applied to Security and the out of trajectory component is applied to Mason.

Next, we say that containers themselves are classified into types. Different types have different rules in terms of how trajectories interact with them.

A *physical container* interacts without any limits. Both out of and into components of trajectories succeed.

A *information container* does not allow the out of components of trajectories to succeed. This does not cause the entire trajectory to fail, only the out of component.

A *source container* does not allow into and out of components of trajectories to succeed. This does not cause the entire trajectory to fail, only the particular component of the trajectory dealing with the source container.

A container which does not allow into components and only allows out of components is rather impractical, as it would simply be empty.

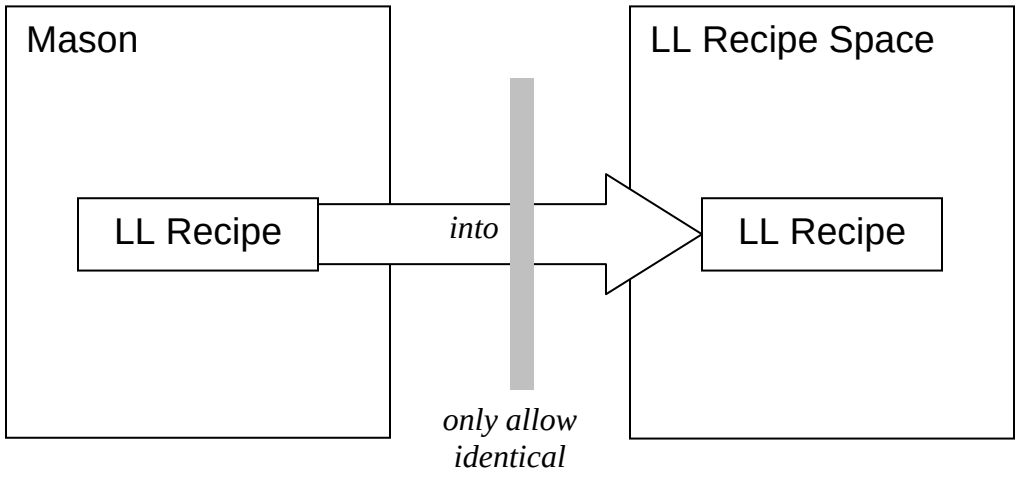
Mason and Security are both information containers. Most of the containers we will be dealing with in this research are information containers. An example of a source container would be an immutable database; speaking in a legal context this might be the precedent for a case: actors within the case generally cannot change precedent because of *stare decisis* (assuming the case is not at the Supreme Court level). Even though out of component trajectories fail, the corresponding into component trajectory would succeed, meaning that other containers could read information within the source container.

We arrive then at a key distinction of container trajectories relative to Jackendoff's trajectories: container trajectories have two components: implicit and explicit, and interact with two containers, the primary and secondary. Containers have three types: physical, information and source. A container's type dictates whether trajectory components succeed in interacting with it.

2.4. Filters

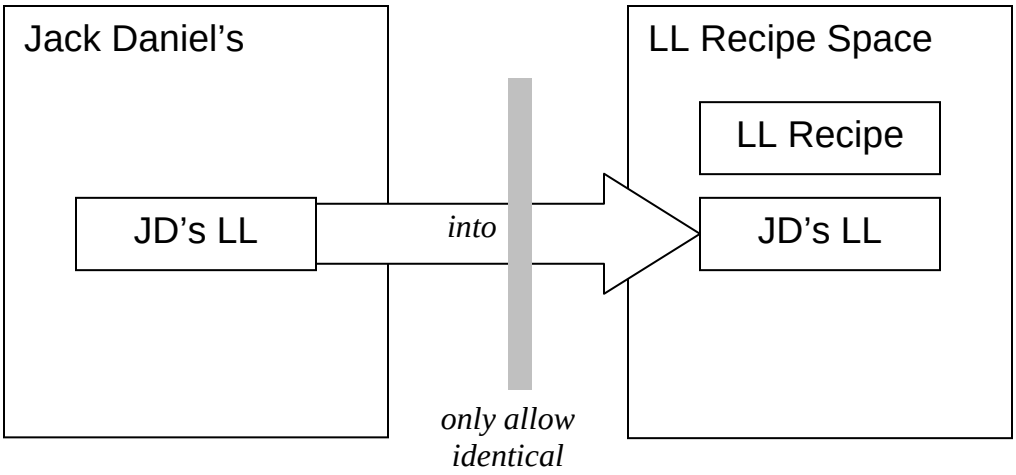
We've established how to represent the first case factor, Security-Measures, as a container trajectory. To represent the next factor, Unique-Product, we'll need to introduce a new concept: filters.

Here's the container trajectory for the case factor Unique-Product in the Mason case:

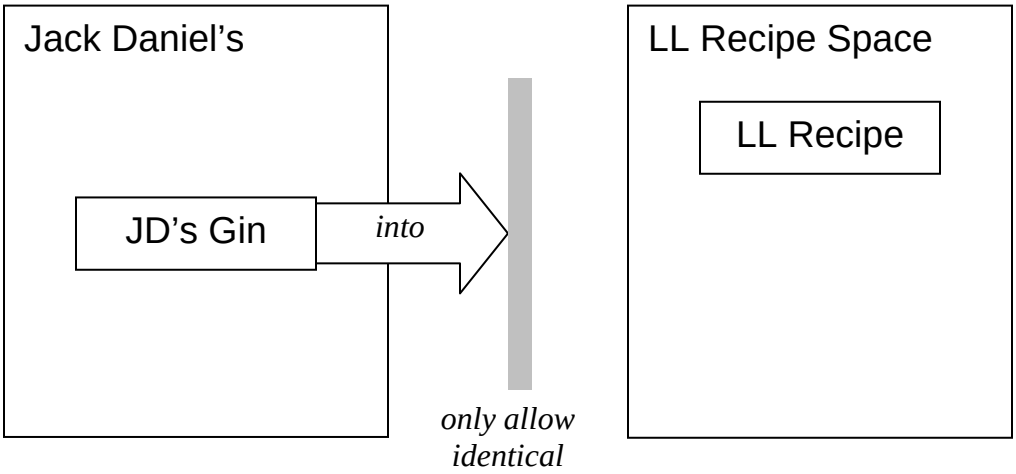


This should look familiar. The LL Recipe container is moving along the *into* trajectory into the LL Recipe Space container. This is a container created to represent the space of products akin to the LL Recipe. Finally, there's a thick gray line over the into trajectory. This is a *filter* on the trajectory. The filter's rule is under the filter itself; here it is *only allow identical*. That means that only new objects identical to the objects already in the container are allowed in the container. If there are no objects in the container, then the first object put in will be the one by which all others are judged. Objects of course can be containers themselves and filter rules can be as complex as desired. In Toulmin's terminology, a filter's rule can be viewed as a qualifier.

How then can we represent why Mason argues that Jack Daniel's product is an Identical Product? The container trajectory for the Identical-Product factor looks like this:

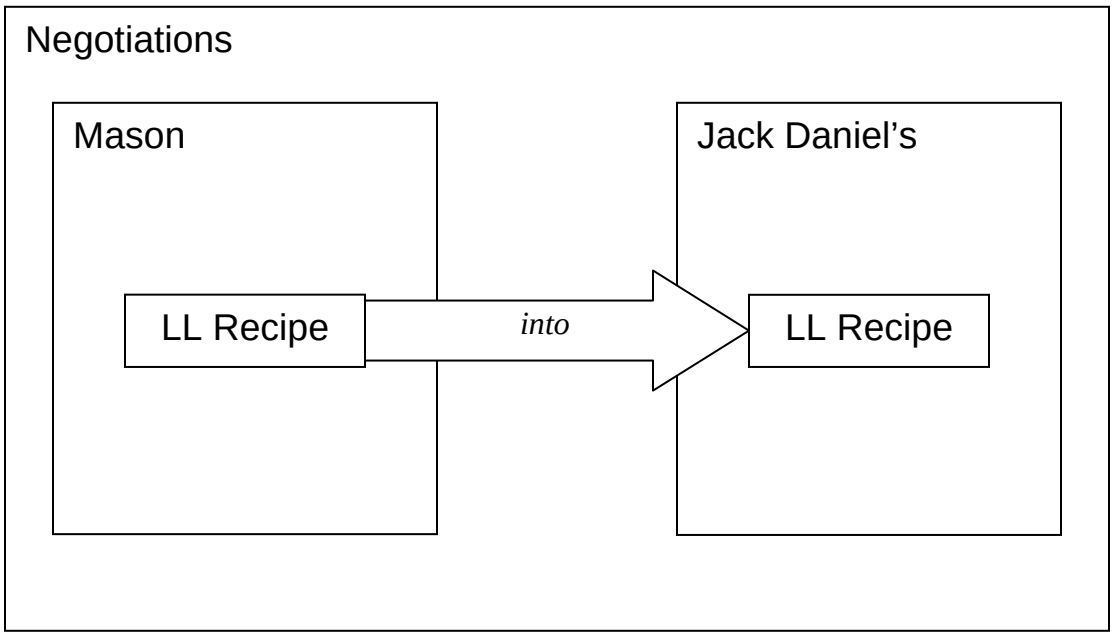


Here we can see that Jack Daniel's Lynchburg Lemonade (JD's LL) is moving along the into trajectory from Jack Daniel's into the LL Recipe Space. It gets past the filter because it is identical to Mason's LL Recipe. If it was not identical, then the container trajectory would look like this (say for JD's Gin), with the filter stopping the trajectory from succeeding:



2.5. Conduits

Next, let's tackle expressing the case factor Disclosure-In-Negotiations as a container trajectory:

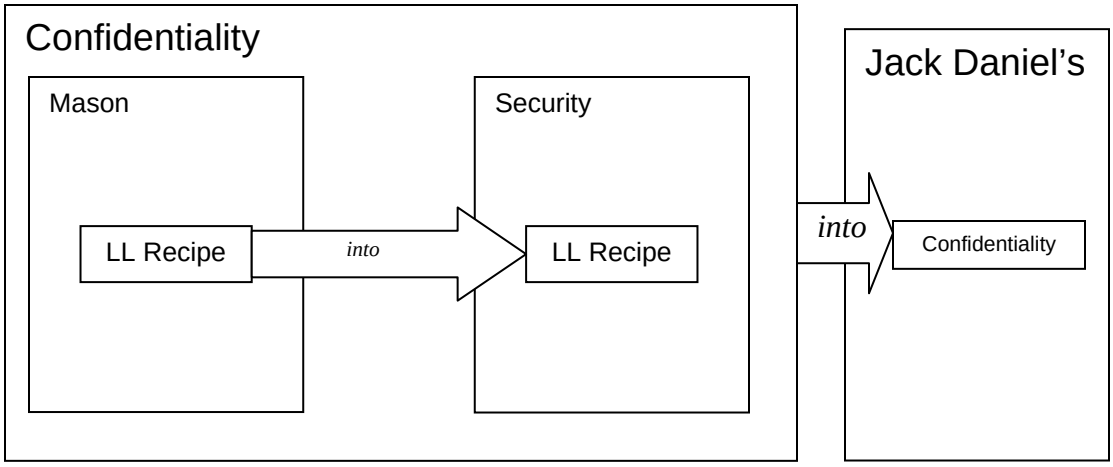


The LL Recipe travels the into trajectory to Jack Daniel's. All of this takes place within the *conduit* of Negotiations. A conduit is the passage a trajectory takes; in other words, for the into trajectory to occur, it had to go through the conduit of Negotiations.

In this diagram, the disclosure (the into trajectory itself) of the LL Recipe by Mason to Jack Daniel's takes place via Negotiations (the conduit). Thus we have represented the Disclosure-In-Negotiations case factor.

Conduits themselves are containers. Conduits are the only containers that can directly contain trajectories; all other containers can only directly contain other containers. Conduits, in contrast, can only contain container trajectories.

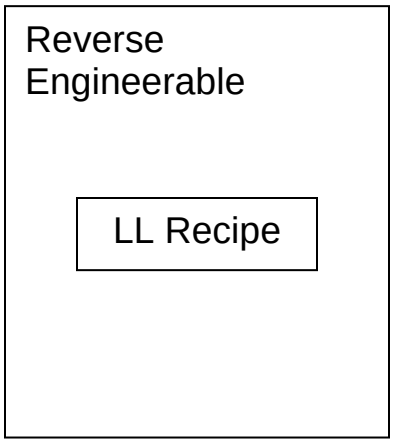
This property allows us to represent the next factor, Knew-Info-Confidential, in the following manner:



Here we have the entire Confidentiality conduit, itself a container, moving along the into trajectory into the Jack Daniel's Container. This represents that Jack Daniel's knew that Mason had taken security measures to product the LL Recipe. Also, notice how container trajectories allow for interaction between case factors in a manner that wasn't possible with case factors alone. Before, one case factor did not matter to another case factor. Now, with container trajectories, case factors can depend on and interact with one another.

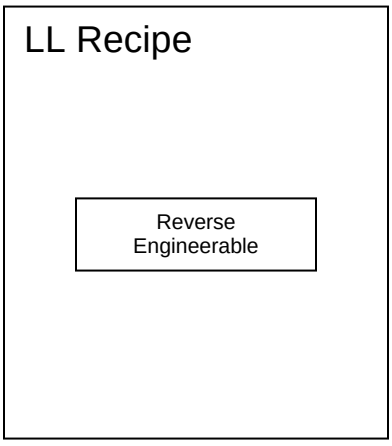
2.6. State and Invertible Containers

What if we don't want to express a trajectory per se and would simply like to express state? We can use containers to do this as well. We simply set up a container representing the state desired and then place any objects that are in that state within the container. Here's an example to represent the Info-Reverse-Engineerable case factor from the Mason case:



The Reverse Engineerable container holds the LL Recipe object, to indicate that the LL Recipe object can be reverse engineered.

Non-conduit containers can be inverted, however. Here is the Info-Reverse-Engineerable from another perspective:



Inverting a container is akin to the concept of trajectory components. Just as an *into* trajectory has an implicit out of trajectory component, so too are containers invertible. The first representation of Info-Reverse-Engineerable can be read as "among the objects that are reverse-engineerable, there lies the LL Recipe. The second representation is "within the LL Recipe, it has a property of being reverse-engineerable."

3. Definitions

3.1. Containers and Objects

An object is an entity within a particular representation. Objects are empty containers, or containers whose contents are beyond the current depth of resolution. Containers are objects that can contain other objects or containers.

Containers are represented by a box with the container's name within the top left corner of the box.

3.2. Trajectories

A trajectory is a representational bridge between two containers. A trajectory is one of these five basic concepts of motion: through, into, out of, toward, and away.

Trajectories are represented by arrows connecting two containers, with the trajectory's name within the arrow.

3.3. Trajectory Components

Trajectories are built up of complementary components: the explicit and implicit component. The explicit component is the one corresponding to a trajectory's name, for the into trajectory, the into trajectory component is the explicit component. The implicit component is the complementary of the trajectory's name, for the into trajectory, the implicit component is the out of trajectory component. Each trajectory component corresponds to one of the two containers that the trajectory bridges.

3.4. Trajectories and Containers

Trajectories bridge two containers. The container corresponding to the destination of the explicit trajectory component is known as the primary reference. The other container, corresponding to the implicit trajectory component, is called the secondary reference.

3.5. Types of Containers

A container can limit whether or not it allows trajectory components of a certain type to succeed. There are three types of containers:

A *physical container* interacts without any limits. Both out of and into components of trajectories succeed.

A *information container* does not allow the out of components of trajectories to succeed. This does not cause the entire trajectory to fail, only the out of component.

A *source container* does not allow into and out of components of trajectories to succeed. This does not cause the entire trajectory to fail, only the particular component of the trajectory dealing with the source container.

3.6. Filters

Whereas container types control the success of trajectory components, filters control the success of entire trajectories. A filter is applied to particular trajectory, and if the filter's criteria are met, the trajectory succeeds. If the filter's criteria are not met, the trajectory fails.

Filters are represented by thick gray lines across trajectory arrows, with the corresponding filter criteria (known as the filter's rule) centered below the bottom of the gray line.

3.7. Conduits

Conduits are special types of containers that can only contain full container trajectories: two containers and the associated trajectory.

Conduits are represented by a large box containing the two containers and the trajectory arrow, with the name of the conduit written in the top left corner of the box.

3.8. Invertible Containers

All non-conduit containers are invertible, meaning that their contents can become their container.

4. Contributions

This paper presents the concept of *container trajectories*, as applied in the context of trade secrets law. Container trajectories are an upgrade of Jackendoff's trajectories that work in information environments. Container trajectories, applied to case factors, provide law with a robust representation for expressing key legal principles. Developing container trajectories provides a valuable target for natural language processing systems; one can imagine, for example, a natural language processing system capable of reading case briefs, forming container trajectories, deriving case factors and automatically providing a set of relevant cases to support either the plaintiff or defendant. Overall, container trajectories are a novel representation for expressing the interaction of information for legal contexts and beyond.

5. Acknowledgements

Many thanks to my project supervisor, Prof. Patrick Winston: he introduced me to this particular field of study as well as taught me how to understand artificial intelligence papers thanks to his Human Intelligence Enterprise Class. Thanks also to Prof. Leslie Kaelbling who encouraged me to pursue research that I found exciting and for her support and guidance throughout my years at MIT. Thanks too to Anne Hunter for her friendship since I first visited MIT and for her help navigating the administrative aspects of MIT. Finally, thanks to my advisor, Prof. Piotr Indyk, for his support and guidance.

6. References

[1] E. L. Rissland, K. D. Ashley, and R. P. Loui, “AI and Law: A fruitful synergy,” Artificial Intelligence, vol. 150, no. 1-2, Nov., pp. 1-15, 2003.

[2] Jackendoff, Ray, Semantics and Cognition, Cambridge, MA: MIT Press, 1983.

[3] K. D. Ashley, and E. L. Rissland, “Law, learning and representation,” Artificial Intelligence, vol. 150, no. 1-2, Nov., pp. 17-58, 2003.

[4] Toulmin, Stephen, The Uses of Argument, Cambridge, UK: Cambridge University Press, 1958.