Transforming Natural Language Stateless Commercial Contracts into Computer Computable Contracts

Transforming Natural Language Stateless Commercial Contracts into Computer Computable Contracts

Pedro Felipe Toledo Correa

A Thesis in the Field of Software Engineering

for the Degree of Master of Liberal Arts in Extension Studies

Harvard University

May 2022

Abstract

Since millennia, contracts have fulfilled a fundamental role to regulate the relation between entities. A contract can be abstracted as an agreement that defines rights and obligations between two or more parties; in this context, the calculation of the resulting rights and obligations, provided the agreement considerations, has fundamentally been a human task given the typical natural language nature of these agreements.

This thesis analyses the context in which the outcome of a contract can be computer calculated and the validity of the calculated result, while proposing an algorithmic approach to transform a natural language contract, within the subset of contracts denominated "Stateless Commercial Contracts", into a computer computable form compatible with tools already available within the legal industry.

Acknowledgements

I would like to thank my thesis director Professor Daniel Martin Katz for listening and supporting the development of this thesis idea, to the people that helped me to complete the set of challenges that allowed me to reach this point in my academic life, and to all the people that supported me with the understanding of the challenge that it was to reach this landmark in my life.

Contents

Chapter 1. Introduction

In the world of Legal Technologies (LegalTech) substantial progress has been achieved in the last few years. Latest advancements on computing power and computing techniques have pushed the boundary of what can be achieved in this area.

The LegalTech industry currently presents different areas in which Natural Language Processing (NLP) is being applied. The work of Robert Dale (Dale, 2019) has classified them in 5 main areas:

- Legal research: finding information relevant to a legal decision
- Electronic discovery: determining the relevance of documents to an information request
- Contract review: checking that a contract is complete and avoids risk
- Document automation: generating routine legal documents
- Legal advise: using question-and-answer dialogues to provide tailored advice

The application of NLP for legal purposes is not a new area but one with many challenges. While it is possible to consider that a legal text has to define unambiguously the relation between entities, with a clear and well defined set of rules, this derives into long complex interdependent sentences; therefore, while most of the information required to analyze a contract resides within the text, there are frequent internal references and applicability conditions.

The development of NLP tools for legal purposes, or their extension from NLP tools (Manning et al., 2014), has been subject of specific analysis in the past. As an example, it is possible to see that there are specific open source tools available like LexNLP (Bommarito et al., 2018) with functionalities that range from simple document segmentation to extraction of entities and tagging; nevertheless, while exploring the current state of the art, it has not been possible to find tools that focus on identifying the mathematical relations defined within a document. Still, there are studies that focus on the identification of meta-data contained within a contract in useful ways: The work of Sleimi et al. (Sleimi et al., 2018) has focused on the extraction of specific meta-data, and the work of Mehta et Al. (Mehta & Kumar, 2015) has pointed to the parsing of semantic clauses.

In order to introduce the problematic presented by this thesis, it is required to start by broadly defining the concept of Contract. In a simplified way, it is possible to propose a general understanding for the concept of "Contract" as "Private Law", a set of rules that specifies rights and obligations to be assigned to the involved parties, agreement that once agreed corresponds to an enforceable compromise. From a practical point of view, the agreement can only be enforceable within the wider context provided by the society: typically the government or another form of power that can assure that the agreement will be fulfilled.

While LegalTech is an interesting intersection between emerging technologies and the ancient concept of law, there is little development at academic level considering the potential market size. This thesis wants to contribute to this area by focusing on a specific topic as it is the "Contract Computability" problem and, within this problem, specifically on how a natural language written contracts could be transformed into a computer computable form.

To introduce the problematic, and while a most precise definition will be pro-

vided further in this thesis, it is possible to summarize the problematic of Contract Computation as the challenge in which, provided the required data, it is possible to define the rights and obligations that will be granted by a contract after an event occurs. It is not possible to conceive a contract without the capability of calculating the outcome, and given the natural language specification of these agreements the contract computation has historically been a human activity.

During the last years, and with the progress of digital technologies, new ways for contract computation have emerged. Concepts as "Smart Contracts" or natural language contracts that has been transformed into computer computational forms have introduced innovations in the field; nevertheless, the frontier between natural language contracts and computer computable contracts has mostly been an unexplored area.

From a practical point of view, it is logical to assume that any entity involved in a contract should have been able to participate in the writing of this agreement, in a way that allows the agreement to cover what the parties want to specify; at the same time, it is logical to consider that, once established, all parties should be independently able to calculate what the outcome of the contract will be under the same set of circumstances. Is in this point where disagreements about the contract outcome can appear; therefore, it becomes necessary the existence of an entity that has the authority to resolve these eventual multiple interpretations such as arbitrators and judges.

As it has been previously mentioned, while different LegalTech areas have been developed for the last decades, the problem of computer contract computation has mostly gotten a calculation approach, this is, the development of tools that allow to write and calculate a contract in a computer compatible manner under the use of specific languages, with specific calculation engines. Independently from this already

solved problem, this thesis aims to the interpretation of natural language written contracts in a way that they can be transformed into a computer computable language that allows a computer calculation for the contract outcome.

While this thesis focuses only on a very narrow type of contracts, the impact of this thesis can be relevant on a mostly unexplored area. The integration of natural language contracts with the capability of computer computing can be considered at the edge of the concept of Natural Language Understanding, this is, a machine that cannot only identify the semantic meaning of a word within a text, but that is also capable to generate an interpretation of what the text specifies.

The applications for this idea are very broad. From this algorithmic proposal, together with the theoretical background, it is possible to project multiple applications specially in highly regulated markets, as commercial contracts for exchange of goods and services, financial agreements and insurance (Borselli, 2020).

This thesis covers from the general concept of contract, in a theoretical framework, up to the proposal of an algorithmic approach to transform a natural language stateless commercial contract into a form that is computer computable with currently existing tools.

Chapter 2. Background

2.1. Contracts

In order to provide a starting point, lets consider the following definition of Contract:

> *A promise enforceable by law (von Mehren, 2019)*

And in a more precise definition, this thesis will consider:

> *A contract is a legally binding document between at least two parties that defines and governs the rights and duties of the parties to an agreement (Ryan, 2006, p.1)*

This definition relies on the concept of agreement and, from it, the definition can be analyzed in its different parts. For the agreement definition we can rely on the following two meanings (Merriam-Webster, 2021, agreement):

- 1a) Harmony of opinion, action, or character
- 2a) An arrangement as to a course of action

At this point, it is possible to understand that a contract is a common understanding, therefore, there should be at least 2 entities that agree upon this common

understanding projecting the possible future contract outcomes when specified events occur.

From the previously stated starting point, it is required to extend the given definition by narrowing down the meaning of its adjectives, substantives and verbs, specifying the characteristics that make an agreement to fall into the specific the category of contract:

- Legally binding: The agreement must be enforceable by law, therefore, it relies on a legal environment in which it can be enforced
- Document: It is written[1]
- At least two parties: A contract requires multiple parties otherwise has no purpose
- Defines and governs: Not only specifies aspects of an agreement, it exercises authority over the agreement
- Rights: Things that can be asked and that must be provided
- Duties: Things that have an obligation of being provided

From an historical point of view, contract law is the product of a business civilization. It will not be found, in any significant degree, in noncommercial societies. In an economy based on barter, most transactions are self-enforcing because the transaction is complete on both sides at the same moment. Even when transactions do not take the form of barter, noncommercial societies continue to work with notions of property rather than of promise. A true law of contracts — that is, of enforceable promises — implies the development of a market economy (von Mehren, 2019).

[1]This can be questionable in the sense that there are legislations that will enforce contracts that are not written

From these definitions and historical relevance it is possible to propose that, while used in multiple different fields, all contracts have, fundamentally, the same fundamental purpose and from it there is a common set of rules that broadly regulate their interpretation. Associated to this, it is possible to establish that contracts are intended to be unambiguous to interpretation in a way that allows the previously stated "harmony of opinion".

2.1.1 Contracts as a logical machine

It has been seen that a contract corresponds to the formality of an enforceable agreement that define the relation between parties. Nowadays, when a contract is written it will be divided in *Clauses*, these clauses will consider environment values, define variables, assign value to variables and define the rights and duties that corresponds to the effects of the agreement. While this broad analysis corresponds to a simplification of a very complex field, as proposed by the different works by Wesley Hohfeld (Hohfeld, 1917) and further analysis of his works (Markovich, 2020), this general approach is a fundamental simplification to propose that the interpretation of the clauses must be calculable and it is possible to see that this calculation occurs triggered by specific events defined within the contract.

The previous analysis allows to propose that a contract can be modeled as a state machine. This approach can be found in the literature as in the work of Flood & Goodenough (2015): This is a very interesting example of how a financial agreement can be represented as a state machine, and from this representation it is possible to propose a way to calculate the agreement outcome. Certainly, the complexity of this state machine representation is not simple and its graph complexity can grow significantly with just a few clauses and conditions, but solving these difficulties is a technical challenge and not theoretical restriction.

The vision that a natural language contract can be equivalent to a logical machine allows to propose the abstraction that any natural language contract can be transformed into a computer computable form. In this area, the work of (Surden, 2012) presents a relevant extended analysis.

2.1.2 Contract computability

Since the beginning of this thesis the concept of "Contract Computability" has been used widely without a clear definition. To the extent of this thesis, contract computability corresponds to the idea that the effects of a contract can be calculated. As it has been previously stated, it is logical to assume that every contract can be computed, otherwise, the rules that define the relation between parties cannot be applied; nevertheless, it is possible to think about a possible computing uncertainty that arises when the need of third parties is required to compute a contract, for example, in cases where the contract specifies that an specialist or an arbitrator must provide an opinion.

In the sense that it is expected that all signing parties should understand the contract, if a third party is required to compute the contract outcome, the contract remains incomputable for the parties (because they lack the third party opinion), but if the contract computability only relies on one specialist, and that specialist is capable to understand all the non specialist parts, it is possible to assume that the contract will be computable for the specialist while not computable for the signing parties. If a contract requires the opinion of more than one independent third party, the contract will not be able to be computed independently for any of them, neither the signers nor the specialists. As final case, it is possible to see that while a contract might require any number of specialists to compute the contract outcome, the signers will require those specialists only when their opinion is required according to the

contract restrictions. If the events that have occurred and that have triggered a contract computation do not require a third party opinion, the contract will remain computable for the signers.

With the previous discussion it is possible to see that the contract computability relies on the capability, for the entity interested into performing the contract computation, to access all the information that is required to compute the contract effects. With this, the concept of contract computability will be abstracted to the possibility of calculating the effect of a contract with an algorithmic, reliable and replicable procedure.

It has been presented that it is fundamental for a contract to be computable, and now it is also possible to assume that a contract is computable in the same way that a state machine is computable. At this point it is necessary to specify some frontier between contract computability and computer contract computability. It has been stated that, by its nature, all contract must be computable, but a computer contract computation is equivalent to a "single party interpretation" and, therefore, will rely on the access to all the required information and the computing capability of processing all of this information to predict a contract effect. To the extent of this thesis it will be defined that a contract is "fully computable" if a machine is capable to compute the contract effect in all possible scenarios and "partially computable" if it is capable of calculating the contract effects only for a part of the full set of possible contract scenarios. As it will be stated further, no contract can be fully computable, therefore, the use of the concept of contract computability will be understood under this theoretical restriction.

2.1.3 Stateless and stateful contracts

The concepts of stateless and stateful are broad definitions that specify the existence or not of a state within a process. A state can be defined as a value that is stored and modified and that is intrinsic to the process that contains it. Lets take as example a light switch: it can be ON or OFF, therefore, a boolean variable. The process of turning ON the switch is stateless, because it doesn't mater in which state the switch is, after turning the switch ON the light status will be ON. The same happens to the process of turning the switch OFF, because while the process has a previous state it doesn't matters. But what about "toggling"? In this case, the result of the process will depend on the current switch state. With the previous example, it is possible to say that the process of turning ON or OFF a switch is a stateless process, but toggling the switch is a stateful process.

Talking about light switches to explain contracts can be a far approximation, but lets take a look to the following two clauses:

Each time the client requires a product, the product must be delivered to the warehouse A (1)

Each time the client requires a product, the product must be delivered to a warehouse different than the last shipment (2)

Here it is possible to see that no prior information is required to comply with the clause 1, but it is not possible to comply with the clause 2 unless it is known to which warehouse the last shipment was; with this, it is possible to state that a clause can be stateless or stateful. For this example, clause 1 is stateless and 2 is stateful.

As an extension of the previous analysis, lets analyze the following two clauses:

The monthly payment will be adjusted by the CPI variation on January and July (3)

The monthly payment will be updated to the original amount corrected by the CPI variation since the beginning of the contract each January and July (4)

What is interesting about these 2 clauses, something we can typically see in a lease contract, is that for the clause 3 it is not possible to calculate the payment amount without knowing what the current payment is. It is possible to calculate the current payment by calculating the previous payment and, if that payment is after the first 6 months of the contract, it is possible to calculate the previous one too. This will imply a chain of calculations so the current monthly payment information is available to enforce the clause; but this is nothing different to having a stored state for the current monthly payment. With this, we can say that the clause 3 is a stateful clause.

From an effect point of view, clause 4 provides the same effect as clause 3; nevertheless, while it establishes how to calculate the monthly payment amount corrected by the CPI variation, it doesn't require to know the current payment amount. This clause just relies on the contract information and the external source of information to be calculated; therefore, this clause can be considered stateless.

For the most precise reader, it is possible to see a difference on the computation data required by the two clauses: the CPI information required to calculate the clause 3 is different from the information required to calculate clause 4, but it is possible to assume that the effort to obtain the CPI variation for the last 6 months is no different from the effort required to calculate the CPI variation for any period.

Given the proposed classification of the clauses as stateless or stateful, it is

possible to extend the concept from clause to contract. If a contract can be computed without the knowledge of any previous contract calculation the contract is stateless, while if a previous calculation effect is required it will be considered stateful; nevertheless, the stateful or stateless nature of a contract will be obtained from the clauses. If any of the contract clauses is stateful, that contract will be stateful, and only if all the contract clauses are stateless the contract will be considered stateless.

At this point it is possible to go back to the clauses 3 and 4 analysis to propose that a stateful contract can have an equivalent stateless contract if all its stateful clauses can be modified to be stateless. The relevance of working with stateful or stateless contracts, from a computibility point of view, is that the computation of a stateful contract requires to know the contract state at the moment of computation or the ability to calculate the current contract state.

It has been observed, on different commercial contracts, that the clauses comply with the stateful/stateless equivalence. Given the computable nature of any contract, linked to the intent of the parties to well define a set of rules, it is not expected that a clause like clause 2 will appear on a commercial contract; nevertheless, there is nothing that prohibit for it to happen.

As a final analysis, it is possible to add an extra precision about statelessness of statefulness. It has been established that a stateless contract is a contract where all the clauses are stateless, but what about a contract where all the clauses are stateless except by one that only gets executed on a very specific case? From this it is possible to say that a contract is strictly stateless if all the clauses are stateless, and we will say that a contract is mildly stateless if the great part of regular calculations is stateless.

2.1.4 Contract incompleteness

From the beginning of this thesis the analysis have been conducted under the assumption that contracts are fully computable agreements in the sense that, provided enough information, the contract contains all the definitions and rules to calculate its effects; nevertheless, in reality this is not the case.

In a broad sense, contracts are inherently incomplete because it is impossible to define all possible aspects of an agreement within cost constraints. It is possible to ask specialists to define an effect for every possible condition that can be conceived but, without considering the costs, it is not possible to define the things that a specialist does not conceive as possible. By other side, contracts are within a legal context, and this legal context can affect the contract outcome because the contract computation could relay on rules that are not within the contract. Finally, even in the impossible case that every single condition gets specified, there could always be a *force majeure* event that affects how the contract rules must be applied.

From a language perspective, a contract can also be incomplete because of the problem of different interpretations. As the language is based on a common understanding about what a sentence means and; therefore, how a rule should be enforced, the concept of "harmony of oppinion" can only reach certain extent and only in a few cases (like quantities) there can be an unquestionable agreement about what is specified.

Regardless of the different contract incompleteness cases presented before, there is also a voluntary incompleteness that can be defined by the parties. The signers of a contract can agree that certain areas of the contract will be explicitly incomplete. The reasons behind this decision can multiple, like the cost of defining an improbable scenario, the perspective about a future re-negotiation, the existence

of regulatory legislation, etc. This type of incompleteness is normally referred as "Incompleteness by design".

In abstract: It is possible to state that contracts are inherently incomplete because of the impossibility to account for all possible scenarios, the existence of external legislation, the possibility of a *force majeure* event, because of differences on the language interpretation and because it can be voluntarily incomplete as part of the contract design.

The problematic about contract incompleteness and its study is recognized as a fundamental part about the study of contracts, to the extent that, in 2016, the work of Oliver Hart about contract incompleteness (Hart, 2017; Hart & Moore, 1999, 2007) got awarded the Nobel Price in Economics.

2.1.5 Computability and incompleteness

The section 2.1 presented a structured approach about the problematic of computing a contract outcome. From its legal definition, it was stated the possibility of reshaping a natural language contract into a logical machine, then it was discussed the theoretical capability of performing an algorithmic contract computation and then the problematic was extended by analyzing some of the complexities of performing the contract computation. In this context, the problematic of contract incompleteness carries the consequence that a contract cannot be fully computable; then, if the intent is to be able to computer compute a contract effect it is required to reduce the computability scope to a safe area where the completeness requirement still holds. For this, the following restrictions must be considered:

- Specificityness: The contract computation can only be performed over specified conditions and effects

- Exactness: A contract can only be computed while the natural language message that defines the agreement is not susceptible to ambiguity

- Constrainedness: The contract computation can only be valid to the extent that there is no influence of external factors that can affect the contract effect. For example: no consideration of external legislation and no consideration of force majeure events.

- Fullness: The contract computation can only be performed if all the input information required for performing the computation is available. In the case of third party consultation, the consultation must be consider as a calculation input

As a natural language contract cannot always comply with the exactness restriction, it is not possible to calculate a contract effect with complete certainty; nevertheless, it is possible to calculate a "conceivable effect": a calculation result that, while considering the previous stated restrictions, can be assumed as the contract effect within an acceptable margin of uncertainty.

The previously stated restrictions allow to restrict the contract computability to the area where the interpretation of a contract as a logical machine still holds, but with the cost of an uncertainty that is no different from the theoretical uncertainty that comes from the language ambiguity and the intrinsic contract incompleteness that is, at the end, a restriction that humans and machines share when computing a contract.

2.1.6 Commercial contracts

So far it has been discussed the broad concept of contracts, but contracts can regulate a vast amount of different types of agreements. In reality, agreements

will typically be drafted with specific considerations within the agreement context, and this can complicate the problem of contract computability because the required generalization to be able to calculate any contract conceivable effect would require a enormous effort; therefore, this thesis focuses on a specific type of contract as the Commercial Contracts.

It is possible to define Commercial Contracts as the agreement between two o more parties on a commercial issue. These issues can be summarized into agreements for the supply of goods and services and the sharing of resources and/or commercial results.

The foreseen benefits to focus only on this type of contracts are:

- Commercial contracts are less susceptible to language ambiguities provided that they are naturally indented to be precise and specific
- Commercial contracts tend to have a more limited amount of conditionals to define rights and duties, and the flow of conditionals tends to be simpler than in other domains
- Commercial contracts tend to have rules that can easily be converted into mathematical formulas; therefore, more suitable for transformation into a computer computable form

The nature of the last benefit can be easily translatable to an already interestingly explored area as solving mathematical problems directly from the natural text that describes them (Kushman et al., 2014; Hosseini et al., 2014; Shi et al., 2015; Roy & Roth, 2016; Koncel-Kedziorski et al., 2015).

2.1.7 Smart Contracts

When exploring the concept of computer contract computability it is possible to find a direct case of computer computable contracts under the concept of Smart Contracts. While there is no complete consensus for a Smart Contract definition, it is possible to explore a set of characteristics that different authors have considered as fundamental:

- They are intended to automatically execute legally relevant events
- They are expressed in a strict language, for example, as computer programs or transaction protocols
- They are automated, no human intervention is required once the contract is set

From this point, different authors establish different additional requirements for the qualification of a contract as a smart contract:

- The smart contracts should be implemented in an environment that allows its self-execution
- The smart contracts are immutable once implemented
- The smart contracts must be contained in a decentralized block-chain network
- The smart contracts transactions must be traceable
- The smart contracts transactions must be irreversible

Starting from the 3 fundamental points, for the objective of this thesis there is no major relevance for the extra conditions to understand that a Smart Contract

is, in essence, a contract that can be computer computed; nevertheless, the contract is directly created in a computer computable form since the very beginning. It is logical to assume that the agreement implemented on the Smart Contract will be the agreement of the parties that have reached a harmony of opinion through natural language, but there must have been a specialist in the middle capable of expressing that natural language agreement into the smart contract strict language.

While the smart contracts are currently considered a relevant area of research and development, the persistent of regular contracts seems to be based on certain features that will prevent them from disappearing (Lipshaw, 2019).

2.2. Technologies

2.2.1 Natural Language

In neuropsychology, linguistics, and the philosophy of language, a natural language or ordinary language is any language that has evolved naturally in humans through use and repetition without conscious planning or premeditation. Natural languages can take different forms, such as speech or singing. They are distinguished from constructed and formal languages such as those used to program computers or to study logic (Lyons, 1991, p 68-70).

Basically, when a person wants to communicate certain information, this information gets enclosed into a message by the use of the set of rules defined by the language in use, then, the message can be communicated to another person with the ability of using the knowledge about the language in order to decode the message to obtain its enclosed information.

In the context of this thesis, the concept of "harmony of opinion" stated on the section 2.1 implies that a group of parties will concur into an agreement (information)

that needs to be enclosed by the rules of a natural language to get transformed into a message that will constitute the contract; therefore, we can propose that a contract written in natural language is the message that encloses the agreement.

With this distinction beteween information and message, it is easier to confront the different concepts related to the algorithmic approaches for the extraction of the information of messages. The following sections present the concepts of natural language processing and natural language understanding from a broad point of view but refined for the problematic of this thesis.

2.2.1.1 Natural Language Processing

As stated in the introduction, the LegalTech industry presents different areas in which Natural Language Processing is being applied; nevertheless, no precise definition of NLP has been provided given the direct semantic interpretation that can be inferred from the name. The challenge to present a specific definition for Natural Language Processing relies on the multiple interpretations that the concept has depending on the field of study. This multiplicity of interpretations has 2 main areas of conflict: where NLP gets classified in the current tree of knowledge and what NLP covers. The first problematic is of no real relevance to the discussion of this thesis; nevertheless, it is necessary to have a clear idea about what NLP covers. To solve this issue, lets consider the following definitions:

- NLP corresponds to the application of computational techniques to the analysis and synthesis of natural language and speech[2]
- NLP focuses on the tokenization of data – the parsing of human language into its elemental pieces[3]

[2]Oxford Languages
[3]IBM - `https://www.ibm.com/watson/natural-language-processing`

- NLP is the use of machine learning to reveal the structure and meaning of text[4]

- NLP is concerned with the interactions between computers and human language, in particular how to program computers to process and analyze large amounts of natural language data[5]

When the different chosen definitions - and this is a very small set of possible definitions - get analyzed and abstracted to their most fundamental parts, it is possible to notice the common element of "language" as the concept of "message", in any form, that humans naturally use to communicate information. Then, the "processing" part of the definitions presents a multiplicity of views about how and in which extent the message can be transformed back into the information it is expected to contain.

To the extent of this thesis, it will be considered that "Natural Language Processing" corresponds the algorithmic ability to extract the information enclosed by humans in a message.

2.2.1.2 Natural Language Understanding

While the concept of NLP focuses on the activity of information extraction from messages, the concept of Natural Language Understating (NLU) focuses on how an algorithmic approach can deal with this information. Different authors propose NLU as a subtopic within the concept of NLP but, given the definition adopted for the extent of this thesis, it will be considered that NLU corresponds to an algorithm that can use the information contained in the natural language message to answer questions about the enclosed information.

Lets take the following message:

[4]Google - `https://cloud.google.com/learn/what-is-natural-language-processing`
[5]Wikipedia - `https://en.wikipedia.org/wiki/Natural_language_processing`

Field	Meta-data	Value
Subject	Entity	Harvard University
Legal status	Classification	Private University
Type	Classification	Research University
Association	Classification	Ivy League
Location	Location	Cambridge, Massachusetts
Foundation	Date	1636
Benefactors	Entity	John Harvard
Characteristics	Classification	Oldest institution of higher education in the United States
Characteristics	Classification	Prestigious at world level

Table 2.1: Harvard University Wikipedia entry information

Field	Meta-data	Value
Subject	Entity	John Harvard
Title	Classification	Cleric
Religion	Classification	English Protestant
Legacy	Description	First benefactor of Harvard University

Table 2.2: John Harvard extracted information from Harvard University Wikipedia entry

> *Harvard University is a private Ivy League research university in Cambridge, Massachusetts. Founded in 1636 as Harvard College and named for its first benefactor, the Puritan clergyman John Harvard, it is the oldest institution of higher learning in the United States and among the most prestigious in the world.*[6]

A simple NLP analysis should allow us to fill the tables 2.1 and 2.2. From this, we can start asking some simple questions:

- In which year was Harvard University founded?
- What is the legal status of Harvard University?

[6] `https://en.wikipedia.org/wiki/Harvard_University`

- Where is Harvard University located?
- Is Harvard University a public institution?
- Was Harvard University founded in the XVII century?

All these questions can be directly answered from the tables, provided there is an algorithm that can associate the question with the field where the information is, and that it is also possible to compare the table retrieved information with the question information, as it is required to answer the last 2 questions. Lets try now the following questions:

- Was John Harvard President of Harvard University?
- How many Research Universities there are in Massachusetts?

The answer to the first question is no, John Harvard was never President of Harvard University, but we cannot answer this from the message information, because answering the question would require specific information related to John Harvard (as extracted on table 2.2) being or not being the President, would require an exhaustive list of all the positions John Harvard held in his life, or it would require an exhaustive list of all the Presidents of Harvard University to analyze if John Harvard is on the list or not. For this question a right answer should be "there is no enough information to answer the question".

For the second question, by 2022 there are 14 Research Universities in the State of Massachusetts, but this answer cannot be obtained from the provided message information. Again, it is possible to take the approach of answering "there is no enough information to answer the question", but it is also possible to answer "there is at least 1 Research University in the State of Massachusetts". It is questionable

if the second answer is or not an answer to the original question, because it is an answer more specifically related to the question "How many research universities can be estimated in the state of Massachusetts?", and the ability of transforming the original question into this secondary question (for which we have some level of information) is the result of a NLU of the question itself.

With the previous analysis, to the extent of this thesis, it will be considered that NLP comprehends the algorithmic approach to obtain information from natural language messages, while NLU comprehends the algorithmic approach to use the information provided by an NLP analysis to provide responses to questions which answers do not simply correspond to the extraction of NLP recovered information.

From this concept is is possible to extend this analysis in the context of the law with the work of Agarwal et al. (2016) that proposes a perspective to be able to go towards machine-understandable contracts

2.2.1.3 NLP and NLU in practice

Lets analyze the following clause:

The seller will charge the buyer 10$USD per unit of product A requested by the buyer if the request involves less than 100 units, and charge 8$USD if the request considers more than 100 units. (5)

The NLP analysis should be able to provide the prices for the goods being exchanged and the conditions in which the price is valid; then, lets consider the following questions:

- What is the total amount the buyer must pay if the buyer requests 42 units of product A?

- What is the total amount the buyer must pay if the buyer requests 126 units of product A?
- What is the total amount the buyer must pay if the buyer requests 42 units of product B?
- What is the total amount the buyer must pay if the buyer requests 100 units of product A?

Now, the NLU analysis should be able to use the information extracted by the NLP analysis to draw the requested conclusions. In these case the conclusions would be:

- The buyer must pay 420$USD
- The buyer must pay 1008$USD
- There is no information to calculate the amount to pay
- There is no information to calculate the amount to pay

The first and second questions get a simple straight answer considering there is the information about the amount of goods and the price per unit for each amount; nevertheless, for the third and fourth questions it is not possible to calculate the amount because, while it is clear the amount and product requested, it is not possible to match the input parameters with the restrictions provided for the know prices. In the first case this is because the product B was never part of the agreement, in the second case this is because while prices for product A are defined, the conditions define the price for buying less than 100 units and for more than 100 units, there is no information for exactly 100 units.

2.2.2 Accord Project

The Accord Project[7] is a non-profit, collaborative, initiative developing an ecosystem and open source tools for smart legal contracts. The Accord Project provides domain specific functionalities, meaning it is purposely designed and engineered for building and running commercial agreements, not generic applications. The Accord Project establishes and maintains a technology neutral foundation for smart legal contracts. The goal is to provide universal technology and tooling that:

- Introduces a common format for smart agreements, reducing the need to adopt and learn different technologies and futureproofing templates
- Enables sharing and reuse of agreement templates
- Facilitates the use of executable agreements across any infrastructure: cloud, blockchain, IoT
- Interoperates with any distributed ledger and blockchain platform, today and in the future

It is possible to appreciate that the Accord Project extends the capability of computer computing contracts with several integrations that allows an interoperativity with other systems, and while this is of no relevance for the focus of this thesis it is a relevant consideration for future developments.

The Accord approach consists in 3 sub-projects: Cicero, Concerto and Ergo. The project Cicero allows the creation of reusable machine readable natural-language contracts and clauses, project Cicero utilizes project Concerto to specify the contract variables that may be bound by the project Ergo that corresponds to a programming

[7] `https://accordproject.org/about`

language specifically engineered for legal agreements. With this, an Accord contract is composed by 3 elements: Text, Model and Logic, each one of them managed by a specific tool.

2.2.2.1 Project Cicero

Project Cicero allows to link natural language contracts with computer computable logic. It works by providing a markup language that can be used to import values from natural language text or to generate natural language text from previously stored values. The markup language provides the contract Grammar as in the following example:

> *The price of each product unit will be* {{*pricenormal*}} *$USD unless the number of products requested within the month is greater than* {{*volume*}}*, in this case the price per unit of product will be* {{*pricevolume*}} *$USD.*

Basically, by enclosing words within double curly braces the markup indicates which are the words that have a special significance from a computing point of view. From this point there are 2 possibilities: Provide a sample text with values or to provide the values for the generation of a sample text. In the first case, the following text:

> *The price of each product unit will be 7 $USD unless the number of products requested within the month is greater than 42, in this case the price per unit of product will be 6 $USD.*

Provides a "Contract Data" in the following form:

```
{
    "$class": "org.accordproject.example.exampleContract",
    "pricenormal": 7,
    "volume": 42,
```

```
    "pricevolume": 6,
    "contractId": "27493e6f-a3b5-4065-93bf-925401e44fbd",
    "$identifier": "27493e6f-a3b5-4065-93bf-925401e44fbd"
}
```

Or vice-versa, this is, by modifying the contract data it is possible to regenerate the sample text.

2.2.2.2 Project Concerto

Project concerto allows for the definition of the data model that will be used for the contract. Logically, the names that are used in the data model correspond to the names used for the Cicero text. For this example, the proposed data model is:

```
namespace org.accordproject.example

import org.accordproject.contract.* from https://models.accordproject.org/
    accordproject/contract.cto
import org.accordproject.runtime.* from https://models.accordproject.org/
    accordproject/runtime.cto

asset exampleContract extends Contract {
    o Integer volume
    o Integer pricenormal
    o Integer pricevolume
}

transaction exampleRequest {
    o Integer request
}

transaction exampleResponse {
```

```
    o Integer payment
}
```

It can be seen that there is a exampleContract "asset" that holds the variables that have been defined in the contract text, and then there are the exampleRequest and exampleResponse "transactions" that define the data structure required to calculate the contract outcome and the data structure to return the contract outcome.

2.2.2.3 Project Ergo

With the text of Project Cicero and the data models from Project Concerto, the last piece is to describe the contract logic through the Ergo language. For this example the proposed logic is:

```
namespace org.accordproject.example

contract example over exampleContract {
        clause exampleClause(request : exampleRequest) : exampleResponse {
                if request.request > contract.volume then
                return exampleResponse {
                payment: request.request * contract.pricevolume
                }
        else
                return exampleResponse {
                payment: request.request * contract.pricenormal
                }
        }
}
```

The project Ergo includes the computing engine for the contract description. To use it, it is first required to initialize the contract state, this is required because project Ergo is also capable of computing stateful contracts. Given that in this

example there is no status, the contract state gets initialized in the following way:

```
{
    "$class": "org.accordproject.runtime.State",
    "$identifier": "4266f45d-b7b4-4554-9f56-4d93a2113cc3"
}
```

Where the identifier is a random string for identifiying the initialization. With this, the code is ready to process a request that needs to be defined in the following way:

```
{
    "$class": "org.accordproject.example.exampleRequest",
    "request": 100
}
```

Following the data structure that was provided on the data model. Finally, the calculation provides the following result:

```
{
    "$class": "org.accordproject.example.exampleResponse",
    "payment": 600,
    "$timestamp": "XXXX-XX-XXTXX:XX:XX.XXX-XX:XX"
}
```

Providing the expected payment for the requested volume of products and a timestamp for the instant of calculation. To test the code lest try another requested quantity:

```
{
    "$class": "org.accordproject.example.exampleRequest",
    "request": 40
}
```

This will trigger the price without volume discount for the payment calculation:

```
{
    "$class": "org.accordproject.example.exampleResponse",
    "payment": 280,
    "$timestamp": "XXXX-XX-XXTXX:XX:XX.XXX-XX:XX"
}
```

Together with the previous mention to storing a contract state, Ergo also allows the definition of functions, to enforce conditions, to separate the logic in clauses, to process dates and times as well as emitting contract obligations in a different way than a contract response, information that can be further use to extend the contract effects with integrations that can execute the obligations.

Chapter 3. Problem statement

This thesis covers the problematic of been able to transform a stateless commercial contract in a way that a computer can calculate the conceivable contract outcome, when an event occurs, by proposing an algorithmic approach for this challenge.

The concepts of "stateless contract" and "commercial contract" to the extent of this thesis have been defined in the previous chapter.

3.1. Research problems

3.1.1 To propose a set of steps required to analyze a natural language contract to extract its information

As stated in 2.2.1, it is possible to abstract that a natural language communication, as a contract, encodes certain information. The first research problem for this thesis is to propose an algorithmic approach that can extract this information.

3.1.2 To propose a set of steps to transform a contract information into a computer computable form

Once the natural language contract information has been extracted, the following research problem corresponds to transform this information into a computer computable form.

3.1.3 To propose an integration that allows for the computer contract computation

Independently of the capability of transforming a contract into a computer computable form, there is still the challenge of executing the computation. There are several tools that allow the calculation of a computer computable contract outcome and, for this, one of the research problems is to provide a practical overview about how this can be implemented.

Chapter 4. Solution

After the supporting background research, this thesis proposes an algorithmic approach for the challenge of transforming natural language stateless commercial contracts into computable contracts. Fundamentally, the input of the algorithm will correspond to a natural language message and the output will correspond to a computer program.

4.1. Natural language statements classification

In general it is possible to abstract a contract as a series of statements. These statements can be of three types: declarative statements, performance statements or conditional statements. Lets see the following clause[1]:

> *{{Company A}}, with a business address at {{Address A}} ("Client"), and {{Company B}}, with a business address at {{Address B}} ("Provider"), enter into this Business Contract for the performance of services as set forth in the statement of work attached to and made part of this Agreement, from time to time as Exhibits, on the following terms and conditions:* (6)

This is a "declarative statement" (DS) because while containing relevant contract information it does not includes any conditional restriction and it does not

[1]Based on PandaDoc templates `https://www.pandadoc.com/`

explicitly states rights nor duties. From this clause it is possible to extract only declarations:

- "Company A" has business address at "Address A"
- "Company B" has business address at "Address B"
- "Client" is a variable that represents "Company A"
- "Provider" is a variable that represents "Company B"
- The contract sets the framework for the relation between "Company A" and "Company B"
- The contract defines the conditions for performance of another describing document named "statement of work"

From this analysis it is possible to see that while there is relevant information it does not set any specific right nor duty (it can be assumed that it sets the global duty for both companies about complying with the contract itself, but this is the nature of the document).

For the second type of statement lets see the following clause[2]:

[2]Based on PandaDoc templates `https://www.pandadoc.com/`

In consideration of the purchase and sale of the property, the Parties have agreed to the following payment amounts. All deposits for this business sale Agreement should be made on {{*Date*}}.

Total purchase price inclusive of all furnishings, fixtures and equipment: {{*Purchase Price*}}. (7)

The purchase price of the business is expressed as a sale of the Assets of the business. It is not assessed as a price per asset but is an overall purchase price for all of the Assets.

This is a Performance Statement (PS) because it specifies an action that must be performed. In this kind of statement it is possible to find mathematical relations, for example:

Entity A must pay B amount each time that entity C does D (8)

That can be presented as:

$$\text{payment\_from\_A} = B * \text{performances\_of\_D\_by\_C} \tag{4.1}$$

By other side, a Conditional Statement (CS) takes the form[3]:

If any acts that are beyond either party's control interfere with the completion of the listed services, the Service Provider shall be entitled to payment for services rendered up to that point in time. No further losses or damages will be granted to either party. (9)

In this case it is simple to identify that under the case that "any acts that are

[3]Based on PandaDoc templates `https://www.pandadoc.com/`

beyond either party's control interfere with the completion of the listed services", the triggering condition, the Service Provider obtains the right of "be entitled to payment for services rendered up to that point in time" while both parties obtain the right of "No further losses or damages will be granted to either party".

In general, it is possible to abstract a conditional statement as:

If {{*Condition*}} *then* {{*List of rights and duties if Condition is True*}} *else* {{*List of rights and duties if Condition is False*}}

A fundamental logic structure for computation; from this, lets simplify the text to increase the complexity analysis with the following clause:

If {{*A amount*}} *is less than* {{*B amount*}}*, the* {{*C product price*}} *is* {{*D*}}*, in other case the* {{*C product price*}} *is* {{*E*}}. (10)

In this case, it is possible to identify how a condition joins 2 PSs. This can be made explicit by putting some parenthesis (in square brackets the condition and in parenthesis the CS):

If [A amount is less than B amount], the (C product price is D), in other case the (C product price is E) (11)

Nevertheless, this simplification hides certain implicit information that a reader must know to predict the contract outcome. A more complete form of the previous CS could be:

If [A amount is less than B amount], the (amount to pay is A times D currency), in other case the (amount to pay is A times E currency) (12)

This is because the context for the statement interpretation is added by using

the commercial concept of "price". Basically, it is logical to assume that when a price is set for a product, and independently that we can have a specific variable value, what is important is the amount to pay. With the previous example, it is possible to propose the following equivalence in an strict language as Python:

```
price = None
if a_amount < b_amount:
    price = d_value
else
    price = e_value
payment = price * a_amount
```

Starting from this DS, PS and CS statements classification, it is also required to consider that contracts are normally separated into clauses, being a clause an assemble of sentences and or paragraphs. A clause provides a particular text separation that mainly serves for reference of DSs, CSs and PSs and for the isolation of variables. Lest see the following example:

> *First: The Company A sells the products B and C. The company will provide these products to Company D for prices depending on the amount of products requested. Company D will perform a single payment for all the requested products.*
> *Second: The product B has a price of E currency per unity, unless the amount requested is over F amount where the price per unit will be G currency.*
> *Third: The product C has a price of H currency, unless the amount requested is less than I amount, in which case the product C will take the same price as product B as stated on clause second.*

(13)

Here it is possible to see that it doesn't really matter the clause separation from a computing point if view. The first clause just provides a context statement that has no enforceable effects. The separation of second and third is more relevant, considering that the word "price" is used in both clauses to set the prices of two different products. But it is only required for "price" to get a unique identifier (e.g: price_1 and price_2) to solve the uniqueness problem. Together with this, the third clause makes a reference to the second clause to re-use a previously declared computation, but with another depending variable. With the previous analysis it is possible to propose the following equivalence in an strict language as Python:

```
payment_b = None
payment_c = None
if (requested_b > amount_f):
    payment_b = price_e * requested_b
else:
    payment_b = price_g * requested_b
if (requested_c < amount_i):
    if (requested_c > amount_f):
        payment_c = price_e * requested_c
    else:
        payment_c = price_g * requested_c
    payment_c = price_h * requested_c
payment = payment_b + payment_c
```

This code might seem sub-optimal because it repeats the programming of conditional statements; nevertheless, by analyzing the reference of clause third to second it is possible to take a different approach with the use of a "function". An equivalent code for the previous program could be:

```
def price_b_calculation(requested_b):
    if (requested_b > amount_f):
```

```
        return price_e
    return price_g

def price_c_calculation(requested_c):
    if (requested_c < amount_i):
        return price_b_calculation(requested_c)
    return price_h

payment = requested_b * price_b_calculation(requested_b) +
          requested_c * price_c_calculation(requested_c)
```

This is certainly a programming optimization that can be achieved to get the same computation under a different code; nevertheless, it is not straight forward from the clauses text and, therefore, it will be considered a code optimization beyond the scope of this thesis.

4.2. Algorithmic approach steps

With the previous definitions, the algorithmic approach to transform a natural language contract into a computer computable will be structured as the following series of steps:

(a) Separating the full text into clauses and each clause into paragraphs and each paragraph into phrases

(b) Classifying the sentences into different types of statements

(c) Parsing the statements for computer computability

(d) Transforming the contract information into a computer strict language that allows computation

For the computer computable form of DSs, CSs and PSs it is required the use of an strict language. For this it has been chosen the Ergo language that is part of the Accord Project environment and, from this, the computing engine will be the Accord Project Environment.

Chapter 5. Elucidation and Results

Based on the previous overview for the solution, this chapter describes the different steps together with the proposed approach to implement each step and an example for how the step would work. This elucidation for the proposed algorithmic approach seeks to provide a fully structured solution for a future software implementation for the procedure.

To work through this proposal, the following example contract will be used[1]:

Food Service Contract

This food service contract is by and between Wayne Foods, the Provider, and LexCorp, the Client. The contract is entered into as of January 1st 1990 and shall remain in effect until December 31st 2020 unless canceled pursuant to the terms listed below.

WHEREAS the Client wishes to engage the Provider as a food service vendor at LexCorp Headquarters, the Facility, and the Provider is willing to provide such services, both parties agree to the terms described herein:

Scope of Services

[1]Based on PandaDoc templates `https://www.pandadoc.com/`

The Provider shall provide staffing for the Client's food service area at the Facility. Staffing shall include all front and back end positions, including cashiers, chefs, and associated support roles but excluding facility roles such as janitors or maintenance positions.

The Provider shall be fully responsible for ordering and maintaining necessary food service supplies, preparing and selling food to customers, and performing basic sanitation such as sweeping, mopping, and dish washing.

The Client shall be responsible for providing all necessary facilities and equipment, for major housekeeping such as waxing floors, disinfecting, cleaning vent hoods, and cleaning grease traps, as well as for providing or performing any necessary facility or equipment maintenance.

Payment

The Provider shall invoice the Client on a monthly basis. Each invoice shall include line items for every product sold at the facility during the preceding month. The products and prices for the execution of this agreement shall be the following:

- Aquaman Sushi Roll: $20.00 per roll

- Batman Burrito: $2.00 per burrito, if more than 100 burritos are sold, each burrito will cost $1.75

- Wonder Woman Footlong Club: $10.00 per club, with a free Club each 42 units sold

- Quinn Soda: $20.00 per gallon, must be provided in 12oz and 24oz cups

Right to Terminate

The Provider shall have the right to terminate this food service contract

by providing written notice to the Client for the following reasons:

- Failure to pay by Client

- Inadequate Facilities

- Client or Provider Bankruptcy

Should the Provider cancel this food service contract for any of the above reasons, the contract shall be considered terminated with cause in accordance with applicable law.

The Client shall have the right to cancel this food service contract by providing written notice to the Provider for the following reasons:

- Inadequate Staffing

- Health Code Violations

- Failure to properly use and maintain facilities in accordance with the listed scope of services

- Failure to provide adequate food services to customers

Likewise, should the Client cancel this food service contract for any of the above reasons, the contract shall be considered terminated with cause in accordance with applicable law.

Contract Renewal

This food service contract must be renewed prior to its expiration date in order to ensure the Provider continues to provide food services and staffing at the Facility. Failure to renew the contract prior to expiration may lead to a lapse in services.

Licensing

The Provider shall be solely responsible for procuring and maintaining all licenses, permits, and authorizations required of a food service provider by law.

5.1. Separating the full text into clauses and each clause into paragraphs and each paragraph into phrases

From the example contract, a first view could confront the reader with a very complicated task to automate, and to simplify this problem the logic approach is to divide an conquer.

To start the procedure it is required to divide the full contract into a set of paragraphs composed by phrases, each of them simple enough to be able to be processed by NLP algorithms, but each of them complex enough to carry enough information for the NLP algorithms to gather enough contextual information to work. Together with this, unnecessary symbols must be removed.

5.1.1 Step description

The full text must be subdivided into paragraphs numbered sequentially, and each paragraph must be subdivided in phrases according to the presence of "periods" and numbered sequentially.

Given the presence of titles, these must disappear but provide a label to a group of paragraphs[2].

5.1.2 Results

Lets take paragraphs 8 to 13 from our example contract:

...

Payment

[2]It must be noted that being able to detect a title from a simple "one sentence" text has its challenges by its own

The Provider shall invoice the Client on a monthly basis. Each invoice shall include line items for every product sold at the facility during the preceding month. The products and prices for the execution of this agreement shall be the following:

- Aquaman Sushi Roll: $20.00 per roll

- Batman Burrito: $2.00 per burrito, if more than 100 burritos are sold, each burrito will cost $1.75

- Wonder Woman Footlong Club: $10.00 per club, with a free Club each 42 units sold

- Harley Quinn Soda: $20.00 per gallon, must be provided in 12oz and 24oz cups

...

To present the results, a JSON format object will be provided for the sentence splitting:

```
{
    ...
    9: {
        1: {
            text: "The Provider shall invoice the Client on a monthly basis"
        },
        2: {
            text: "Each invoice shall include line items for every product sold at
     the facility during the preceding month"
        },
        3: {
            text: "The products and prices for the execution of this agreement
    shall be the following"
```

```
        }
    },
    10: {
        1: {
            text: "Aquaman Sushi Roll: \$20.00 per roll"
        }
    },
    11: {
        1: {
            text: "Batman Burrito: \$2.00 per burrito, if more than 100 burritos
    are sold, each burrito will cost \$1.75"
        }
    },
    12: {
        1: {
            text: "Wonder Woman Footlong Club: \$10.00 per club, with a free Club
    each 42 units sold"
        }
    },
    13: {
        1: {
            text: "Harley Quinn Soda: \$20.00 per gallon, must be provided in 12oz
     and 24oz cups"
        }
    },
    ...
}
```

An another-one will be provided for the detected structure after removing the titles:

```
{
```

```
    section: "Food Service Contract",
    content: [
        2,
        3,
        {
            section: "Scope of Services",
            content: [
                5,
                6,
                7
            ]
        },
        {
            section: "Payment",
            content: [
                9,
                10,
                11,
                12,
                13
            ]
        }
        ...
    ]
}
```

5.2. Classifying the sentences into different types of statement

As seen on section 4.1, it is proposed that statements are Declarative (DS), Performance (PS) or Conditional (CS). This base classification is related to the impact that these different effects have on the contract computation. A DS will assign values

to variables, a PS will define rights and obligations and a CS will define which PS and or DS should be considered based on conditions.

By its nature, a PF specifies something that needs to be performed, but DS and CS are assignations or conditionals, therefore, they will be composed by sub statements. A DS will be composed by a variable that will adopt the value of another statement, and CSs will define under a logical test if one or another statement should be applied. With this, it is possible to see expect as a result a tree of nested statements.

5.2.1 Step description

For each identified statement in the previous step a type of statements needs to be proposed and added to the data structure, then, the CSs and DSs needs to be separated into their component statements recursively until there are only PS on the structure leaves.

5.2.2 Results

From the data structure proposed at 5.1, the extra information is added to add the sentence classification:

```
{
    ...
    9: {
        1: {
            text: "The Provider shall invoice the Client on a monthly basis",
            type: "PS"
        },
        2: {
```

```
        text: "Each invoice shall include line items for every product sold at
 the facility during the preceding month",
        type: "PS"
    },
    3: {
        text: "The products and prices for the execution of this agreement
shall be the following",
        type: "DS",
        variable: "prices",
        value: {
            text: None,
            type: "JS",
            statements: [{
                text: None,
                type: "RS"
                statement: 10
            }, {
                text: None,
                type: "RS"
                statement: 11
            }, {
                text: None,
                type: "RS"
                statement: 12
            }, {
                text: None,
                type: "RS"
                statement: 13
            }]
        }
    }
```

```
},
10: {
    1: {
        text: "Aquaman Sushi Roll: \$20.00 per roll",
        type: "DS",
        variable: "aquamansushiroll_price",
        value: {
            text: "\$20.00 per roll"
            type: "PS"
        }
    }
},
11: {
    1: {
        text: "Batman Burrito: \$2.00 per burrito, if more than 100 burritos
are sold, each burrito will cost \$1.75",
        type: "DS",
        variable: "batmanburrito_price",
        value: {
            text: "\$2.00 per burrito, if more than 100 burritos are sold,
each burrito will cost \$1.75",
            type: "CS"
            test: "more than 100 burritos are sold",
            iftrue: {
                text: "each burrito will cost \$1.75",
                type: "PS"
            },
            iffalse: {
                text: "\$2.00 per burrito",
                type: "PS"
            }
```

```
        }
    }
},
12: {
    1: {
        text: "Wonder Woman Footlong Club: \$10.00 per club, with a free Club
each 42 units sold"
        type: "DS",
        variable: "wonderwomanfootlongclub_price",
        value: {
            text: "\$10.00 per club, with a free Club each 42 units sold",
            type: "JS",
            statements: [{
                text: "\$10.00 per club",
                type: "PS"
            }, {
                text: "with a free Club each 42 units sold",
                type: "CS",
                test: None,
                iftrue: None,
                iffalse: None,
            }]
        }
    }
},
13: {
    1: {
        text: "Harley Quinn Soda: \$20.00 per gallon, must be provided in 12oz
 and 24oz cups",
        type: "DS",
```

```
            variable: "harleyquinnsoda_price",
            value: {
                text: "\$20.00 per gallon, must be provided in 12oz and 24oz cups
    ",
                type: "JS",
                statements: [{
                    text: "\$20.00 per gallon",
                    type: "PS"
                }, {
                    text: "must be provided in 12oz and 24oz cups",
                    type: "PS"
                }]
            }

        }
    },
    ...
}
```

This example shows an interesting case with statements 12 and 13, because it is possible to establish that the text separated by periods (in the previous step) does not only contains one statement but a series of statements separated by commas. In this case, a new type of statement is required to describe this situation. For this it will be defined a "Join Statement" (JS) as a statement composed by more than one independent statement.

Together with statements 12 and 13, there is a special case when there are conditions that must be applied only when another condition is met, an example for this could be appreciate if we add the following clause for our example contract:

Exclusively during the month of December, the prices will be increased

by 20%.

This is a special challenge as this conditional statement does not provides an election of PSs but an effect on related statements 10 to 13, and from here there are three possible approaches: creating a conditional statement for each of statements 10 to 13, to replicate statements 10 to 13 with the price modification or adding a conditional step that needs to be applied after the price calculations were performed.

Under this situation, the approach would be to replicate the statements 10 to 13 with a price modification. This decision is because it is the most direct approach from the natural language context as in statement 9.3 there is the declaration of the group of variables "prices" and the group of variables is the one getting modified by this extra statement.

With this analysis, a statement grouping must be provided. When we analyze the example, there are 3 statements groupings and these grouping are linked to a declarative statement: 10 to 13 are linked to 9 (declaration of products and prices), 16 to 18 are linked to 15 (provider conditions to terminate the contract) and 19 to 22 are linked to 18 (client conditions to terminate the contract). With the previous example, it has also been introduced the concept of Related Statement (RS) to declare from one statement all the related statements that are related and that must be considered as a group.

5.3. Parsing statements for computer computability

With the different statements classified and grouped by the previous step, the following step is to structure the flow of calculations that the contract describes. The concept of "flow of conditions" pretends to describe the set of different paths that allows to conclude the rights and duties granted by the contract.

From the five types of statements: Declarative, Performance, Conditional, Joint and Reference, only the CSs affect the flow of conditions and those are the central analysis for this step; nevertheless, there are also some global considerations like, in our example, the validity of the contract. From paragraph 2 it is possible to read:

> *This food service contract is by and between Wayne Foods, the Provider, and LexCorp, the Client. The contract is entered into as of January 1st 1990 and shall remain in effect until December 31st 2020 unless canceled pursuant to the terms listed below.*

From this we can conclude that the contract is valid from January 1st 1990 until December 31st 2020, unless the following paragraph has been executed:

> *This food service contract must be renewed prior to its expiration date in order to ensure the Provider continues to provide food services and staffing at the Facility. Failure to renew the contract prior to expiration may lead to a lapse in services.*

And unless the contract, while valid for the original period or because of an extension, has been terminated as stated on paragraphs 15 to 25.

This situation presents a problem as the "contract validity" is itself a "state" and this algorithmic approach is presented for "stateless" contracts.

Under this condition there are 2 approaches: simply reject the contract (as it is not stateless) or to consider the state under certain framework. The first approach has no much sense as the contract validity is an intrinsic state for any contract, this algorithmic approach could only be applied to unquestionable valid contracts;

nevertheless, for purposes of the state of "contract validity" it is logical to assume that if there is a request for the contract calculation, the contract is currently valid and, therefore, the initial state is that the contract is valid and we do not encounter the state calculation problem described in 2.1.3. The calculation result could be that the contract is no longer valid, but the calculation must assume that the contract is valid to be able to calculate the outcome.

On parallel to CSs, the DSs and PSs needs to be transformed into a form that identifies the fundamental computation variables and the relation that links them.

5.3.1 Step description

Each CS must be analyzed to identify the variable and conditions that define which PS must be performed. If a CS affects a group of CS, the CS must be inserted into the DS that serves as a header for the group. The DS and CS needs to be shaped into a form that allows its calculation by identifying the variable, values and relations that they define.

5.3.2 Results

From our example we have 2 conditional statements that contain a CS: 11 and 12. The following data structure extension for statement 11:

```
{
    ...
    11: {
        1: {
            text: "Batman Burrito: \$2.00 per burrito, if more than 100 burritos
    are sold, each burrito will cost \$1.75",
            type: "DS",
            variable: "batmanburrito_price",
```

```
            value: {
                text: "\$2.00 per burrito, if more than 100 burritos are sold,
    each burrito will cost \$1.75",
                type: "CS"
                test: "more than 100 burritos are sold",
                variables: [
                    burritos_sold
                ],
                calculation: "burritos_sold > 100",
                iftrue: {
                    text: "each burrito will cost \$1.75",
                    type: "PS",
                    variable: "batmanburrito_price",
                    value: "\$1.75"
                },
                iffalse: {
                    text: "\$2.00 per burrito",
                    type: "PS",
                    variable: "batmanburrito_price",
                    value: "\$2.00"
                }

            }
        }
    },
    ...
}
```

And here we can also see how the PS and DS can be abstracted for computation; nevertheless, a different situation is encountered when we want to extend statement 12:

```
{
    ...
    12: {
        1: {
            text: "Wonder Woman Footlong Club: \$10.00 per club, with a free Club
    each 42 units sold"
            type: "DS",
            variable: "wonderwomanfootlongclub_price",
            value: {
                text: "\$10.00 per club, with a free Club each 42 units sold",
                type: "JS",
                statements: [{
                    text: "\$10.00 per club",
                    type: "PS",
                    variable: "wonderwomanfootlongclub_price",
                    value: "\$10.00"
                }, {
                    text: "with a free Club each 42 units sold",
                    type: "CS",
                    test: None,
                    iftrue: None,
                    iffalse: None,
                }]
            }
        }
    },
    ...
}
```

Here we can identify a CS on 12.1.1 but is is not straightforward how to apply it. For example if "each 42 units there is one Club free" we can establish that when

42 units are sold the total price to pay is $410 and, therefore, the price per Club is $9.7619 (different from the original $10.00) but if the units sold are 43 the price per unit is $9.7674. If we reach 83 Clubs the price per Club is $9.8809, but at 84 Clubs we get back the $9.7619 price.

Another approach could be to establish a discount over the total price, using a language like Python we can specify that this discount can be calculated as `price = 10 * (wonderwomanfootlongclub_sold - wonderwomanfootlongclub_sold // 42)`, and somehow it is necessary to define where this discount needs to be applied.

This algorithmic approach will not cover such cases as the only way to apply these non-linear calculations is by applying procedures that cannot be directly obtained from the simple analysis of the natural language text.

Returning over the extra condition that was considered on the previous section:

> *Exclusively during the month of December, the prices will be increased by 20%.*

We can propose the following interpretation:

```
{
    ...
    X: {
        1: {
            text: "Exclusively during the month of December, the prices will be
    increased by 20\%"
            type: "CS",
            test: "Exclusively during the month of December",
            variables: [month],
            calculation: "month == 12",
            iftrue: {
```

```
                text: "the prices will be increased by 20\%",
                type: "PS",
                variable: "prices",
                value: "prices * 1.2"
            },
            iffalse: None
        }
    },
    ...
}
```

This case present us that there are certain contextual values that need to be provided in order to calculate the contract. Typically these values could be related to date and time, but also locality, weather, etc.

5.4. Transforming the data structure into a strict computer computable language

After the three previous steps we have reached the point where our data structure provides us with a structured version of the information encoded in natural language by the contract text (the message); then, we can finally approach the proposal of a computer computable version of this contract by the use of the Accord Project.

We have seen from 2.2.2 that an Accord Project contract is composed by 4 parts: The marked contract, the marked contract data, the data model, the logic model.

5.4.1 Step description

The generation of the 4 Accord Project parts must be approached by 4 steps:

(a) Contract markup: Each statement must be transformed into a marked version and joined with the title information extracted in step 1

(b) Markup data: The initial values declared in the contract need to be identified and extracted for the marked contract data

(c) Data model: The variables detected in the contract needs to be classified as contract information, contract input and contract output

(d) Logic model: The CSs, PSs, and DSs must be transformed into the Ergo language

5.4.2 Results

5.4.2.1 Contract Markup

Payment

The Provider shall invoice the Client on a monthly basis. Each invoice shall include line items for every product sold at the facility during the preceding month. The products and prices for the execution of this agreement shall be the following:

- Aquaman Sushi Roll: ${{*aquamansushiroll_price_1*}} *per roll*

- Batman Burrito: ${{*batmanburrito_price_1*}} *per burrito, if more than* {{*batmanburrito_sold_limit_1*}} *burritos are sold, each burrito will cost $*{{*batmanburrito_price_2*}}

- Quinn Soda: ${{*quinnsoda_price_1*}} *per gallon, must be provided in 12oz and 24oz cups*

Here we can see some relevant approaches:

- The labels `_price`, `_sold` and `_limit` have been used to describe the nature of the detected quantities
- Each time a quantity appears, a number is added to avoid the reuse of variables

- The cup sizes for Quinn Soda have not been considered variables because while they are numbers they are not part of the contract computation

5.4.2.2 Markup Data

From the previous identification of variables, the following contract data structure can be concluded:

```
{
  "$class": "org.accordproject.empty.EmptyContract",
  "aquamansushiroll_price_1": 20,
  "batmanburrito_price_1": 2,
  "batmanburrito_sold_limit_1": 100,
  "batmanburrito_price_2": 1.75,
  "quinnsoda_price_1": 20,
  "contractId": "e51cfc5e-e7b0-4e31-a13e-ea3dfc5a5476",
  "$identifier": "e51cfc5e-e7b0-4e31-a13e-ea3dfc5a5476"
}
```

5.4.2.3 Data model

With the detected variables, then we can propose the following data model for the contrat section:

```
namespace org.accordproject.empty

import org.accordproject.contract.* from https://models.accordproject.org/
    accordproject/contract.cto
import org.accordproject.runtime.* from https://models.accordproject.org/
    accordproject/runtime.cto

// Template model
asset EmptyContract extends Contract {
```

```
  o Double aquamansushiroll_price_1
  o Double batmanburrito_price_1
  o Integer batmanburrito_sold_limit_1
  o Double batmanburrito_price_2
  o Double quinnsoda_price_1
}

// Request
transaction EmptyRequest extends Request {
  o Integer aquamansushiroll_sold
  o Integer batmanburrito_sold
  o Integer quinnsoda_sold_12oz
  o Integer quinnsoda_sold_24oz
}

// Response
transaction EmptyResponse extends Response {
  o Double payment
}
```

To proceed with the previous transformation there are several considerations that have not been descrived as part of the procedure, as the definition of the required data types that compose the "Contract" variables, the input variables (and their types) that compose the "Request" neither the "Response" variable and type.

A more precise approach at the third stage needs to be defined to gather this information.

5.4.2.4 Logic Model

We can finally propose the following logic model for the section:

```
namespace org.accordproject.empty
```

```
contract Empty over EmptyContract {
  clause donothing(request : EmptyRequest) : EmptyResponse {
    let aquamansushiroll_price_computed = contract.aquamansushiroll_price_1;
    let batmanburrito_price_computed =
        if request.batmanburrito_sold > contract.batmanburrito_sold_limit_1 then
                contract.batmanburrito_price_2
        else
                contract.batmanburrito_price_1;
    let quinnsoda_sold = integerToDouble(request.quinnsoda_sold_12oz*12 + request.
    quinnsoda_sold_24oz*24)/128.0;
    let quinnsoda_price_computed = contract.quinnsoda_price_1;
    return EmptyResponse{
      payment: aquamansushiroll_price_computed*integerToDouble(request.
    aquamansushiroll_sold) + batmanburrito_price_computed*integerToDouble(request.
    batmanburrito_sold) + quinnsoda_price_computed*quinnsoda_sold
    }
  }
}
```

Chapter 6. Conclusions

This thesis has tackled an important problem in the field of Legal Technologies. This work lays the foundation from a theoretical point of view for the possibility of transforming natural language contracts into computer computable contracts, by identifying the context and limitations as well as providing an overall structure for how this problem could be tackled. Many simplifications have been taken to reach this initial approach and, even with them, the challenge appears to be a complex dare.

Working at the intersection of Computer Science and Law is an interesting area full of opportunities. The work presented in this thesis is a novel approach for the practical challenge to the theoretical possibility of computers taking care of computing contracts straight from their natural language form without human intervention, a problem that has not been widely discussed in the academic literature.

To build upon this thesis there are several possible future work areas, for example: the extension of the solution for increasing the coverage of border cases as well as an eventual commercial application for the proposed solution.

References

Agarwal, S., Xu, K., & Moghtader, J. (2016). Toward machine-understandable contracts. *AI4J–Artificial Intelligence for Justice*, (pp.1̃).

Bommarito, M., Katz, D., & Detterman, E. (2018). Lexnlp: Natural language processing and information extraction for legal and regulatory texts. *CoRR*, abs/1806.03688.

Borselli, A. (2020). Smart contracts in insurance: a law and futurology perspective. In *InsurTech: A Legal and Regulatory View* (pp. 101–125). Springer.

Dale, R. (2019). Law and word order: Nlp in legal tech. *Natural Language Engineering*, 25(1), 211–217.

Flood, M. & Goodenough, O. (2015). Contract as automaton: The computational representation of financial agreements. *OFR Working Paper 15-04*.

Hart, O. (2017). Incomplete contracts and control. *American Economic Review*, 107(7), 1731–1752.

Hart, O. & Moore, J. (1999). Foundations of incomplete contracts. *Review of Economic Studies*, 66(1), 115–138.

Hart, O. & Moore, J. (2007). Incomplete contracts and ownership: Some new thoughts. *American Economic Review*, 97(2), 182–186.

Hohfeld, W. N. (1917). Fundamental legal conceptions as applied in judicial reasoning. *The Yale Law Journal*, 26(8), 710–770.

Hosseini, M. J., Hajishirzi, H., Etzioni, O., & Kushman, N. (2014). Learning to solve arithmetic word problems with verb categorization. In *EMNLP*, volume 523533: Citeseer.

Koncel-Kedziorski, R., Hajishirzi, H., Sabharwal, A., Etzioni, O., & Ang, S. D. (2015). Parsing algebraic word problems into equations. *Transactions of the Association for Computational Linguistics*, 3, 585–597.

Kushman, N., Artzi, Y., Zettlemoyer, L., & Barzilay, R. (2014). Learning to automatically solve algebra word problems. In *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)* (pp. 271–281).

Lipshaw, J. M. (2019). The persistence of "dumb" contracts. *Stan. J. Blockchain L. & Pol'y*, 2, 1.

Lyons, J. (1991). *Natural Language and Universal Grammar: Essays in Linguistic Theory*, volume 1. Cambridge University Press.

Manning, C. D., Surdeanu, M., Bauer, J., Finkel, J. R., Bethard, S., & McClosky, D. (2014). The stanford corenlp natural language processing toolkit. In *Proceedings of 52nd annual meeting of the association for computational linguistics: system demonstrations* (pp. 55–60).

Markovich, R. (2020). Understanding hohfeld and formalizing legal rights: the hohfeldian conceptions and their conditional consequences. *Studia Logica*, 108(1), 129–158.

Mehta, S. & Kumar, A. (2015). A novel natural language processing (nlp) based approach for developing automated semantic clause parser. *International Journal of Research in Engineering and Technology*, 04, 22–26.

Merriam-Webster (2021). *Merriam-Webster.com*. Merriam-Webster, Incorporated.

Roy, S. & Roth, D. (2016). Solving general arithmetic word problems. *arXiv preprint arXiv:1608.01413*.

Ryan, F. (2006). *Contract law*. Dublin: Thompson Round Hall.

Shi, S., Wang, Y., Lin, C.-Y., Liu, X., & Rui, Y. (2015). Automatically solving number word problems by semantic parsing and reasoning. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing* (pp. 1132–1142).

Sleimi, A., Sannier, N., Sabetzadeh, M., Briand, L., & Dann, J. (2018). Automated extraction of semantic legal metadata using natural language processing. In *2018 IEEE 26th International Requirements Engineering Conference (RE)* (pp. 124–135).

Surden, H. (2012). Computable contracts. *UCDL Rev.*, 46, 629.

von Mehren, A. (2019). *contract*. Encyclopedia Britannica.