A Formalization of a Startup Finance Transaction Model using Alloy

Rodrigo Ehrlich Stevaux

Advisor: Prof. Ana Cristina Vieira de Melo

Dissertation presented to the Institute of Mathematics and Statistics at the University of São Paulo, as part of the requirements for the degree of Master of Science in Computer Science

São Paulo

2023
Abstract

This dissertation presents a software design of a fragment of the domain of startup financing. Our design can be machine-checked for key safety integrity properties, and is written as a formal specification of the domain.

In the given domain, we start from a data specification language built by an industry coalition. The data specification describes how our domain can be modeled from a syntactic perspective.

We give it semantics by adopting the Alloy formal modeling language and analyzer, aiming at a more expressive and correct model, by capturing invariants in the problem domain. Our model and can be explored and checked automatically by the Alloy Analyzer. We can both express and check for critical invariants of the problem domain.

Our research contributes to the field of formal methods as it shows how one can proceed from a semiformal specification to a formal one, evaluates what the results, and contributes to other researchers and practitioners in the field of formal methods by providing a case study of a real-world problem domain.

Keywords: Formal Specification, Formal Methods, Alloy Analyzer, Bounded-Model Checking, Capitalization Table Management, Venture Capital, Semantic Data Specifications, Domain-Specific Formalisms
Resumo

Essa dissertação apresenta um design de software de um fragmento do domínio de financiamento de startups. Nosso design pode ser verificado por máquina quanto a propriedades chave de integridade de segurança e é escrito como uma especificação formal do domínio.

No domínio em questão, começamos a partir de uma linguagem de especificação de dados construída por uma coalizão da indústria. A especificação de dados descreve como nosso domínio pode ser modelado de uma perspectiva sintática.

Damos a ela semântica adotando a linguagem de modelagem formal Alloy e analisador, visando um modelo mais expressivo e correto, capturando invariantes no domínio do problema. Nosso modelo pode ser explorado e verificado automaticamente pelo Analisador Alloy. Podemos tanto expressar quanto verificar invariantes críticos do domínio do problema.

Nossa pesquisa contribui para o campo dos métodos formais, pois mostra como se pode proceder de uma especificação semiformal para uma formal, avalia os resultados e contribui para outros pesquisadores e praticantes no campo dos métodos formais ao fornecer um estudo de caso de um domínio de problema do mundo real.

I dedicate this thesis to my late, loved mother, and to my father.

I thank my advisor, Prof. Ana Cristina Vieira de Melo, not only for her academic guidance and support throughout this work, but for the friendship and care she has shown me since during this endeavor.

I am grateful to my institution for providing me with the opportunity to pursue this work, and from all I have learned from my colleagues and professors.

I thank all the people who helped me arrive at this point, including my family, friends, and colleagues.
<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>I</td>
<td>Background</td>
<td>14</td>
</tr>
<tr>
<td>1</td>
<td>Introduction</td>
<td>15</td>
</tr>
<tr>
<td>1.1</td>
<td>The economic motive to adopt a specification</td>
<td>15</td>
</tr>
<tr>
<td>1.2</td>
<td>Complexities in managing capitalization tables</td>
<td>16</td>
</tr>
<tr>
<td>1.3</td>
<td>Open-source efforts</td>
<td>17</td>
</tr>
<tr>
<td>1.4</td>
<td>Structure</td>
<td>17</td>
</tr>
<tr>
<td>2</td>
<td>Capitalization tables and the need for specifications</td>
<td>18</td>
</tr>
<tr>
<td>2.1</td>
<td>Overview</td>
<td>18</td>
</tr>
<tr>
<td>2.1.1</td>
<td>Use in startup financing</td>
<td>19</td>
</tr>
<tr>
<td>2.1.2</td>
<td>Capitalization table over two investment rounds</td>
<td>21</td>
</tr>
<tr>
<td>2.2</td>
<td>JSON Schema</td>
<td>23</td>
</tr>
<tr>
<td>2.2.1</td>
<td>Validating the presence of specific keys in a document</td>
<td>24</td>
</tr>
<tr>
<td>2.2.2</td>
<td>Composing schemas to form more complex schemas</td>
<td>24</td>
</tr>
<tr>
<td>2.2.3</td>
<td>Validating numeric, string and array values</td>
<td>25</td>
</tr>
<tr>
<td>2.2.4</td>
<td>Achievements and limitations of JSON Schema</td>
<td>26</td>
</tr>
<tr>
<td>2.3</td>
<td>The Open Cap Table format</td>
<td>27</td>
</tr>
<tr>
<td>2.3.1</td>
<td>File format</td>
<td>27</td>
</tr>
<tr>
<td>2.3.2</td>
<td>The existence of an implied conceptual model within the data model</td>
<td>28</td>
</tr>
<tr>
<td>2.3.3</td>
<td>Organizing principles of the underlying model</td>
<td>28</td>
</tr>
<tr>
<td>2.3.4</td>
<td>Opening example of the complete lifetime of a security</td>
<td>29</td>
</tr>
<tr>
<td>2.3.5</td>
<td>A note on the current folder structure of the OCF</td>
<td>30</td>
</tr>
</tbody>
</table>
## II Models

### 3 A new model for capitalization tables

<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.1</td>
<td>An introduction to Alloy</td>
<td>52</td>
</tr>
<tr>
<td>3.1.1</td>
<td>Parts of an Alloy model</td>
<td>53</td>
</tr>
<tr>
<td>3.2</td>
<td>A brief overview of Alloy related literature</td>
<td>54</td>
</tr>
<tr>
<td>3.3</td>
<td>Motivation</td>
<td>54</td>
</tr>
<tr>
<td>3.4</td>
<td>Relation to the Open Cap Table Format</td>
<td>55</td>
</tr>
<tr>
<td>3.5</td>
<td>Methodology</td>
<td>55</td>
</tr>
<tr>
<td>3.6</td>
<td>Key Concepts in the Model</td>
<td>56</td>
</tr>
<tr>
<td>3.6.1</td>
<td>Treatment of Securities</td>
<td>56</td>
</tr>
<tr>
<td>3.6.2</td>
<td>Treatment of transactions</td>
<td>57</td>
</tr>
<tr>
<td>3.7</td>
<td>Signatures for Securities, Transactions, and Stakeholders</td>
<td>58</td>
</tr>
<tr>
<td>3.7.1</td>
<td>Abstract security signature</td>
<td>58</td>
</tr>
<tr>
<td>3.7.2</td>
<td>Abstract signature for transactions</td>
<td>60</td>
</tr>
<tr>
<td>3.7.3</td>
<td>Concrete transactions signatures</td>
<td>61</td>
</tr>
<tr>
<td>3.7.4</td>
<td>Constraints</td>
<td>77</td>
</tr>
<tr>
<td>3.8</td>
<td>Vesting system</td>
<td>78</td>
</tr>
<tr>
<td>3.8.1</td>
<td>Conditions</td>
<td>79</td>
</tr>
<tr>
<td>3.8.2</td>
<td>Triggers</td>
<td>80</td>
</tr>
<tr>
<td>3.8.3</td>
<td>Unrolled evaluation function</td>
<td>82</td>
</tr>
<tr>
<td>3.9</td>
<td>Checks</td>
<td>83</td>
</tr>
</tbody>
</table>
4 Relation to Contracts and code normativity

4.1 Our Work in the Context of Code Normativity ....................... 94
4.2 Role of Contracts in Investment Activities ............................. 95
4.3 Incompleteness and Reliance on Legal Frameworks ................... 95
4.4 The Challenge for Software Engineers: From Prose to Code .......... 95
4.5 Simplifying Contracts Through Our Model ............................ 95
4.6 Code Normativity and Community Efforts ............................ 96
   4.6.1 Smart Contracts ............................................. 96
   4.6.2 Tax and Accounting: The Catala Approach .................... 96
4.7 An example rendition of our model as prose contracts ............... 97

5 Conclusion ............................ 100

5.1 Future Work .............................................. 101
List of Figures

2.1 Capitalization tables as states of a system ........................................... 20
2.2 Directory structure of the OCF distribution ........................................... 30
2.3 Types in the OCF for general use ............................................................ 31
2.4 Conversion mechanisms ...................................................................... 32
2.5 Conversion rights ...................................................................................... 32
2.6 Vesting events and conditions, periods and portions ............................... 33
2.7 Conversion triggers, divided in elective and automatic .......................... 33
2.8 Enums in the OCF .................................................................................... 34
2.9 Primitive transactions in the OCF ............................................................ 35
2.10 Transactions relevant to stock options and vesting ............................... 36
2.11 Transactions relevant to stock ................................................................. 36
2.12 Transactions relevant to warrants .......................................................... 37
2.13 Transactions relevant to convertibles ...................................................... 37
2.14 Adjustment transactions in the OCF ....................................................... 37
2.15 An issuance ............................................................................................. 39
2.16 A partial cancellation .............................................................................. 40
2.17 A partial transfer .................................................................................... 40
2.18 Vesting schemas in the OCF ................................................................. 44
2.19 The convertible securities system ........................................................... 48

3.1 Metamodel of the abstract security ......................................................... 58
3.2 Metamodel of the transaction signature, including relationships between transactions and securities, and the extensions PlanTx and StockTx ........... 61
3.3 Metamodel of the stock transaction signature, showing the specific transaction types and their share fields ........................................................... 63
| 3.4 | Metamodel of the plan transaction signature, showing the specific transaction types and their share fields | 63 |
| 3.5 | The graph formed by securities and transactions | 64 |
| 3.6 | An example instance of the stock issuance transaction | 65 |
| 3.7 | An example instance of the stock partial cancellation transaction | 67 |
| 3.8 | An example instance of the stock partial transfer transaction | 68 |
| 3.9 | An example instance of the stock total cancellation transaction | 70 |
| 3.10 | An example instance of the stock total transfer transaction | 70 |
| 3.11 | An example instance of the plan issuance transaction | 72 |
| 3.12 | An example instance of the plan cancellation transaction | 73 |
| 3.13 | An example instance of the plan exercise transaction | 73 |
| 3.14 | An example instance of the plan vesting date transaction | 75 |
| 3.15 | An example instance of the plan vesting event transaction | 76 |
| 3.16 | Metamodel of the vesting system | 79 |
List of Tables

2.1 A capitalization table ........................................... 18
2.2 A capitalization table at company formation ..................... 21
2.3 A capitalization table after a seed round .......................... 21
2.4 A capitalization table after an option pool is created .......... 22
2.5 A capitalization table after a Series A investment ............... 22
2.6 A capitalization table after an exit ................................ 23
2.7 Available Validations in JSON Schema (Types and enums) ........ 24
2.8 Composition Operators in JSON Schema .......................... 25
2.9 Available Validations in JSON Schema (Numbers) ................. 25
2.10 Available Validations in JSON Schema (Strings) .................... 26
2.11 Available Validations in JSON Schema (Collections) ............. 26
2.12 Available Validations in JSON Schema (Objects) .................. 26
2.13 Files in the Open Cap Table format ................................ 28
2.14 Key components in the OCF ...................................... 38
2.15 Transactions and their supported security types ................... 43

3.1 Comparison between our model and the original model .......... 55
3.2 Transactions in the model ........................................... 62
### List of Listings

<table>
<thead>
<tr>
<th></th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Primitive transaction schema</td>
<td>41</td>
</tr>
<tr>
<td>2</td>
<td>Transfer schema in the OCF</td>
<td>42</td>
</tr>
<tr>
<td>3</td>
<td>Abridged vesting term schema</td>
<td>45</td>
</tr>
<tr>
<td>4</td>
<td>Abridged vesting condition schema</td>
<td>46</td>
</tr>
<tr>
<td>5</td>
<td>Abstract signature for securities</td>
<td>59</td>
</tr>
<tr>
<td>6</td>
<td>Signature for a stock security</td>
<td>59</td>
</tr>
<tr>
<td>7</td>
<td>Signature for a plan security</td>
<td>60</td>
</tr>
<tr>
<td>8</td>
<td>Abstract signature for a transaction</td>
<td>61</td>
</tr>
<tr>
<td>9</td>
<td>The stock issuance transaction signature</td>
<td>66</td>
</tr>
<tr>
<td>10</td>
<td>The stock partial cancellation transaction signature</td>
<td>67</td>
</tr>
<tr>
<td>11</td>
<td>The stock partial transfer transaction signature</td>
<td>69</td>
</tr>
<tr>
<td>12</td>
<td>The stock total cancellation transaction signature</td>
<td>70</td>
</tr>
<tr>
<td>13</td>
<td>The stock total transfer transaction signature</td>
<td>71</td>
</tr>
<tr>
<td>14</td>
<td>The plan issuance transaction signature</td>
<td>72</td>
</tr>
<tr>
<td>15</td>
<td>The plan cancellation transaction signature</td>
<td>73</td>
</tr>
<tr>
<td>16</td>
<td>The plan exercise transaction signature</td>
<td>74</td>
</tr>
<tr>
<td>17</td>
<td>The plan vesting transaction signature</td>
<td>75</td>
</tr>
<tr>
<td>18</td>
<td>The plan vesting date transaction signature</td>
<td>76</td>
</tr>
<tr>
<td>19</td>
<td>The plan vesting event transaction signature</td>
<td>77</td>
</tr>
<tr>
<td>20</td>
<td>Ordering of parent securities</td>
<td>78</td>
</tr>
<tr>
<td>21</td>
<td>Ordering of parent transactions</td>
<td>78</td>
</tr>
<tr>
<td>22</td>
<td>Signature of a condition.</td>
<td>79</td>
</tr>
<tr>
<td>23</td>
<td>Signature of a trigger.</td>
<td>80</td>
</tr>
<tr>
<td>24</td>
<td>Signature of an AfterEvent trigger.</td>
<td>80</td>
</tr>
<tr>
<td>25</td>
<td>Signature of an AfterDate trigger.</td>
<td>81</td>
</tr>
<tr>
<td>26</td>
<td>Signature of a Conjunction trigger.</td>
<td>81</td>
</tr>
</tbody>
</table>
27 Signature of a \textbf{Disjunction} trigger. ........................................ 81
28 Signature of a \textbf{Negation} trigger. ........................................ 82
29 Unrolled evaluation function. ...................................................... 83
30 Plan security shares ................................................................. 85
31 New shares ................................................................................. 85
32 Cancelled shares ......................................................................... 86
33 Outstanding shares ..................................................................... 86
34 Check plan shares ....................................................................... 87
35 Check cancelled shares ................................................................. 87
36 Check outstanding shares ............................................................. 87
37 Count checks ............................................................................... 89
38 Opening the \texttt{util/graph} module for securities and transactions .... 90
39 Check that the \texttt{parent} relationships induce an acyclical graph. .... 90
40 Checking that all securities contain an issuance in their lineage ....... 90
41 No overlap in portfolios ............................................................... 91
42 Predicate describing a stock option grant ...................................... 98
Glossary

**acquisition** An acquisition is when one company purchases another company. Depending on how many shares are acquired, the acquiring company may gain control of the acquired company. 9, 16

**angel investor** An angel investor is an individual who provides capital for a business start-up, usually in exchange for convertible debt or ownership equity. These investors typically support startups in the early stages of growth. 9

**asset** An asset is something that can eventually generate cashflows. Because not all future cashflows are known with certainty, the value of an asset must be discounted to reflect the risk that those cashflows do not meet expectations. 9, 38

**asset class** An asset class is a group of securities that have similar characteristics. Stocks, bonds, and real estate are all asset classes. 9, 16, 19, 35, 41, 42

**capitalization table** A capitalization table is a table that lists all the securities issued by a company. The capitalization table lists the number of shares issued, the type of security, the price per share, and the date of issuance. The capitalization table is used to calculate the ownership of each shareholder. 9, 15–19, 21, 22, 27, 32, 38, 49, 54, 55

**cliff period** The cliff period is the period of time before which no stock options are vested. After the cliff period, stock options are typically vested monthly. 9, 46

**common stock** Stock that holds no special rights beyond a share in profits. Common stock is the most common type of stock. 9, 16

**convertible debt** In startup financing, it is typical to encounter convertible debt. Convertible debt is a loan that can be converted into equity at a later date. The conversion
is typically triggered by a future financing round. The conversion price is typically set at a discount to the price of the future financing round. Since debt is safer, convertible debt has lower investment risk. Conversion is typically at the option of the holder.. 9, 47

debt Debt is a loan that must be repaid. Companies might raise funds via equity issuances or debt issuances. Debt is issued as security in terms of the amount that was loaned, the interest rate, and the maturity date. Debt is safer than equity, and must be repaid before equity holders can receive any cashflows.. 9, 20, 38, 47

dilution Dilution is the reduction in ownership stake that occurs when new shares are issued. In the special case that in the financing round investors purchase new shares proportionally to their then current stake, no dilution occurs. Founders and employees can also be diluted by the issuance of new shares.. 9

equity Equity represents ownership interest in a company. It can come in the form of stocks or stock options and may be granted to founders, employees, or investors. Equity holders have a claim to the profits of the company.. 9, 20, 21

exercise Stock options are exercised and become stocks. The strike price is the price at which the stock options can be converted to equity. They can only be exercised after they have been vested.. 9, 19, 20, 30, 38, 42

financing round Each stage of financing is also called a financing round. Each financing round is typically led by a lead investor, who sets the terms of the financing round. The terms of the financing round include the valuation of the company, the price per share, and the type of security issued.. 9, 19

initial public offering During an initial public offering, a company sells shares of its stock to the public for the first time. This is also known as going public and is a way for companies to raise capital for new investments or to pay off existing debt.. 9, 16

issuance An issuance is the creation of a new security plural. 9

issuer Companies are issuers of securities.. 9, 27, 29, 30
liquidity event  A liquidity event is an event that allows a company’s stakeholders to cash out or make a profit from their investment. This could be a sale of the company (also known as an exit), an IPO, or a large dividend distribution. 9

preferred stock  Preferred stock is stock that holds special rights. As an example of a special right, preferred stock might have a guaranteed dividend payment but less voting rights.. 9, 16, 21, 22, 29, 42, 47

security  A security is a financial asset that can be bought and sold. Stocks, options and debt notes are all securities. Every security has an Issuer. A loan from a bank is not a security, because the bank can not generally sell the loan to another bank.. 5, 9, 16, 27, 29, 30, 35, 38–40, 42, 44, 47–49, 54

share  Shares represent a fraction of ownership in a company. The number of shares a stock holder owns is the starting point for calculating their ownership stake in the company.. 9

staged financing  Staged financing is a financing strategy in which a company raises funds in stages. The first stage is typically called the seed round, with subsequent stages receiving a latin alphabet letter (such as Series A, Series B, etc.). Staged financing allows investors to reduce their risk by investing in stages, and allows the company to raise funds as it grows.. 9, 20

stakeholder  A stakeholder is any person, legal or natural, with an economic interest in a company, including all debt, option and stock holders.. 9, 16

startup company  A startup company is a new company that is searching for a business model as it grows. Startup companies are typically funded in stages and by specialized venture capital investors such as individual (angel) investors and funds. Startup companies usually aim for high growth and high returns, by choosing projects with higher risk.. 9, 15, 19, 20

stock  Stocks are securities that represent ownership in a company. Stock are typically issued as shares (e.g. 100 shares of Apple stock). Shares are fractions of a company’s total ownership, plural. 9

stock class  A stock class is a group of stocks that have similar characteristics. Common stock and preferred stock are both stock classes.. 9, 16, 30, 37, 42, 47
**stock option** Stock options are securities that give the right for their holder to purchase stock at a predefined price (the strike price) at a predefined date (the maturity date). The value of a stock option is the different between the market price for the stock and the strike price. Stock options are typically issued to employees as part of their compensation package.. 9

**strike price** The strike price is the price at which a stock option can be exercised. The strike price is typically set at the market price of the stock at the time of issuance, but may be further discounted to incentivize employees.. 9, 16

**sweat equity** Startup founders raise capital by selling shares of their stock to investors. For example, an investor might take 20 of 100 shares, leaving founder with 80 shares that were not issued against cash. This complement of the shares that held by investors is called sweat equity.. 9, 21

**transaction** A transaction refers to the issuance, change, transfer and cancellation of securities. A transaction is typically initiated by a stakeholder, and must be approved by the company. Most transactions involve a cost, with money changing hands in the opposite direction of the securities.. 9, 16, 20, 27, 28, 30, 32, 35–39, 42, 49

**venture capital** Venture capital is a form of private equity financing that is provided by venture capital firms to startups and early-stage companies that have been deemed to have high growth potential or which have demonstrated high growth. 9, 15, 16, 19

**vesting period** A vesting period is a period of time during which an employee must remain employed in order to receive the full value of their stock options. Vesting periods are typically 4 years, with a 1 year cliff period. Only vested stock options can be exercised.. 9, 34
Part I

Background
Chapter 1

Introduction

Capitalization tables are essential documents required for conducting venture capital investments in startup companies. A capitalization table depicts the ownership structure of a company, and this structure is subject to change following investments, transfers, and acquisitions.

Errors in capitalization tables can prove costly and may lead to potential legal disputes. Traditionally, capitalization tables have been maintained in spreadsheets, a method which is prone to error and difficult to audit. Furthermore, spreadsheets do not adhere to a standard format, requiring all parties involved in a transaction to agree on a uniform format before exchanging data.

The inherent issues associated with maintaining capitalization tables in spreadsheets have prompted the emergence of several companies that offer capitalization table management as a service. These companies provide a web-based interface for managing capitalization tables, which aims to streamline the process and reduce errors. However, these services are not without limitations. The underlying data models they use are proprietary and not publicly accessible, and the rules for updating the capitalization tables are often not explicitly defined.

1.1 The economic motive to adopt a specification

To give objective measures of cost and risk to the task of validating capitalization tables, we can consider the following:
1. 38,644 venture capital deals were closed in 2021 (according to accounting firm KPMG [1]). Each deal requires tens of hours of lawyers and accountants.

2. The venture capital market invested USD 671 billion from investors in 2021, and exits (including initial public offerings and acquisitions) amounted to USD 1.378 billion (also according to the same KPMG report).

3. As of June 2023, Apple, Google, NVIDIA, Amazon, and Microsoft alone were worth USD 9 trillion. All companies started as venture capital investments. Their combined market up is comparable only to the gross domestic product of the United States of America (USD 23 trillion) and that of China (USD 17.7 trillion).

1.2 Complexities in managing capitalization tables

Capitalization tables are complex to manage for several reasons:

1. Different asset classes. Common stock entitles profit sharing and voting rights, generally proportional to the number of shares. preferred stock generally carries less voting rights but more economic rights, such as having priority over common stock in case of, and can in some cases be exercised (converted) to common stock. Stock options are securities that can be converted to shares of a given stock class upon the payment of a strike price, which, if lower than the market price, will result in profits for the option holder.

2. Distinction between actual and diluted shares.

3. Difference in the price each investor paid for shares, and the respective investment cost must be recorded properly to calculate returns in case the company is acquired or goes public.

4. There is no information on the transactions that led to the current capitalization table, which must be considered to validate a capitalization table.

The last point is the most critical one. The only way to attest that a capitalization table accurately reflects the correct stakes of each stakeholder is to validate the transactions that led to the current capitalization table.

This is a non-trivial task, and is extremely error-prone if done manually, which alone motivates the need for a specification for capitalization tables. As we shall see in depth, a
candidate for such a specification is in development by the Open Cap Table Coalition [2], based on JSON Schema, which we introduce next.

1.3 Open-source efforts

Recently, the Open Cap Table Coalition [2], formed by industry members, has been working on a standard for capitalization tables.

The standard is based on a data model that is publicly available, and that can be used to build software systems that manage capitalization tables. However, the data model is not accompanied by a formal specification of the rules for updating the capitalization tables. This is an unfortunate consequence of the specific semantics of the chosen technology, JSON Schema, which is syntax-oriented and does not allow for the specification of complex rules.

1.4 Structure

In this thesis, we present a formal specification of the rules for updating capitalization tables. We do so by first modeling the data model in Alloy, and then by specifying the rules for updating the capitalization tables in Alloy. We then use the Alloy Analyzer to check that the rules are consistent with the data model, and that the rules are free of errors.

Our goal is to show that formal methods, in the form of Alloy, can be feasibly used to specify the rules for updating capitalization tables. We hope that this work will lead to a clearer understanding of the rules for updating capitalization tables, and to the development of software systems that implement these rules.

The structure of the thesis is as follows:

• In Chapter 2, we present the original data model for capitalization tables, created by the Open Cap Table Coalition, and the JSON Schema technology that is used to specify the data model.

• In Chapter 3, we present our Alloy model of the data model for capitalization tables, and the rules for updating capitalization tables.

• In Chapter 4, we discuss the relation between our model and traditional prose contracts in the context of code normativity

• In Chapter 5, we present our conclusions, and discuss future work.
Chapter 2

Capitalization tables and the need for specifications

2.1 Overview

A capitalization table is a table that shows the ownership stakes in a company. We give an example in table 2.1. The table shows the ownership stakes of the founders, angel investors, funds, and employees. The table also shows the number of shares issued, the type of security, the price per share, and the date of issuance. The capitalization table is used to calculate the ownership of each shareholder.

<table>
<thead>
<tr>
<th>Asset class</th>
<th>Stakeholder</th>
<th>Shares Actual/diluted</th>
<th>Cost</th>
<th>% Ownership Actual/diluted</th>
</tr>
</thead>
<tbody>
<tr>
<td>Common stock</td>
<td>Founders</td>
<td>700/700</td>
<td>0</td>
<td>78%/70%</td>
</tr>
<tr>
<td>Preferred stock</td>
<td>Angel investors</td>
<td>50/50</td>
<td>0</td>
<td>6%/5%</td>
</tr>
<tr>
<td>Preferred stock</td>
<td>Funds</td>
<td>150/150</td>
<td>300</td>
<td>16%/15%</td>
</tr>
<tr>
<td>Options</td>
<td>Employees</td>
<td>0/100</td>
<td>0</td>
<td>0%/10%</td>
</tr>
<tr>
<td>Total</td>
<td></td>
<td>900/1000</td>
<td>350</td>
<td>100%/100%</td>
</tr>
</tbody>
</table>

Table 2.1: A capitalization table
Stakeholders have interests in different asset classes, resulting in some shares and ownership at a given cost. Cost might be different for different investors as they invest in different times, as a company’s value might fluctuate.

Options are granted to employees, who can exercise the options and exchange them for actual shares. As an incentive for the employee to stay with the company, virtually all options have a vesting schedule, which is a period of time that the employee must remain with the company in order to exercise the options. The most typical case is to vest shares after 1 year, and then vest the remaining shares monthly over the next 3 years. Vesting will be the subject of a later section.

2.1.1 Use in startup financing

Capitalization tables are used specially in startup financing. Startup companies are financed in stages, which each stage requiring the achievement of certain objectives and milestones but also providing a larger of capital.

Stages in startup financing

Stages are defined in terms of the phase of the company’s development and the amount of capital required to achieve the next stage. The stages are[3]:

- Seed/Startup financing: relatively small amount of capital, for companies mostly aiming to prove a business concept. Beyond Venture capital firms, individual investors (“angel” investors) are also common sources of seed funding.
- Early-stage financing: moderate amounts of capital, for advancing pilots or productions and searching for commercial viability evidence.
- Mid-stage financing: this stage revolves on supplying a company with working capital so that it can continue to grow.
- Later-stage financing: this stage is for companies that are already on the way to profitability, and serves as a bridge to an initial public offering (IPO) or a sale to a larger company.

Financing rounds are numbered not by their stage but by letters, typically: Seed, Series A, Series B, Series C and so on. Investment sizes range from USD 1 million in seed rounds to USD 100 million or more in later rounds.
In this manner, the computer scientist reader may note that a capitalization table is the state of a system, and financing operations are transitions of that state. Figure 2.1 illustrates this idea.

![Cap table](state)

- Investments, transfers and other transactions (transition)
  - More investments, transfers and other transactions
  - Even more investments, transfers and other transactions

Figure 2.1: Capitalization tables as states of a system

**Financing rounds and investment contracts**

Each staged financing round is defined in contracts that define business rules that are potentially complex to validate. It is common for the investors of each round to have different rights and obligations:

- Seed rounds are frequently financed via a loan to the company that can later on be converted to equity. Precise triggers and conversion rates are defined in a contract. Debt confers more rights to the investor than equity, such as the right to receive interest payments and the right to be paid back before equity holders.

- Startup companies typically reserve a portion of shares for employees, which are typically granted in the form of stock options. Stock options are granted according to vesting schedules, and become shares when the employee exercises the option.

Conversions, transfers, vesting and other events can only happen within certain conditions, and those conditions must always be validated, on the risk of misrepresenting the company’s ownership structure.
2.1.2 Capitalization table over two investment rounds

Since we hinted at seeing the capitalization table as states in a transition system, we can now see an example of a capitalization table over two investment rounds:

**Company formation**

At company formation, the capitalization only reflects the founders’ ownership of the company, by giving them a number of shares with no matching cost. **Equity** that has no matching investment is typically called **Sweat equity**.

<table>
<thead>
<tr>
<th>Asset class</th>
<th>Stakeholder</th>
<th>Shares Actual/diluted</th>
<th>Cost</th>
<th>% Ownership Actual/diluted</th>
</tr>
</thead>
<tbody>
<tr>
<td>Common stock</td>
<td>Founders</td>
<td>100/100</td>
<td>$0</td>
<td>100%/100%</td>
</tr>
<tr>
<td>Total</td>
<td></td>
<td>100/100</td>
<td>$0</td>
<td>100%/100%</td>
</tr>
</tbody>
</table>

Table 2.2: A capitalization table at company formation

After a seed round, a group of investors joined the capitalization table, purchasing 100 shares for $1 each. The founders’ ownership is diluted to 50% of the company. The new shares are issued as **preferred stock**.

<table>
<thead>
<tr>
<th>Asset class</th>
<th>Stakeholder</th>
<th>Shares Actual/diluted</th>
<th>Cost</th>
<th>% Ownership Actual/diluted</th>
</tr>
</thead>
<tbody>
<tr>
<td>Common stock</td>
<td>Founders</td>
<td>100/100</td>
<td>$0</td>
<td>50%/50%</td>
</tr>
<tr>
<td>Preferred stock</td>
<td>Seed investors</td>
<td>100/100</td>
<td>$100</td>
<td>50%/50%</td>
</tr>
<tr>
<td>Total</td>
<td></td>
<td>200/200</td>
<td>$100</td>
<td>100%/100%</td>
</tr>
</tbody>
</table>

Table 2.3: A capitalization table after a seed round

The company creates an option pool for its employees, reserving 20 shares for that purpose. The founders’ ownership is diluted to 45% of the company. The new shares are issued as options.
Table 2.4: A capitalization table after an option pool is created

<table>
<thead>
<tr>
<th>Asset class</th>
<th>Stakeholder</th>
<th>Shares Actual/diluted</th>
<th>Cost</th>
<th>% Ownership Actual/diluted</th>
</tr>
</thead>
<tbody>
<tr>
<td>Common stock</td>
<td>Founders</td>
<td>100/100</td>
<td>$0</td>
<td>45%/45%</td>
</tr>
<tr>
<td>Preferred stock</td>
<td>Seed investors</td>
<td>100/100</td>
<td>$100</td>
<td>45%/45%</td>
</tr>
<tr>
<td>Options</td>
<td>Employees</td>
<td>20/20</td>
<td>$0</td>
<td>10%/10%</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td></td>
<td><strong>220/220</strong></td>
<td><strong>$100</strong></td>
<td><strong>100%/100%</strong></td>
</tr>
</tbody>
</table>

And then the company receives a Series A investment, where a new group of investors purchases 100 shares for $2 each. The founders’ ownership is diluted to 40% of the company. The new shares are issued as **preferred stock**.

Table 2.5: A capitalization table after a Series A investment

<table>
<thead>
<tr>
<th>Asset class</th>
<th>Stakeholder</th>
<th>Shares Actual/diluted</th>
<th>Cost</th>
<th>% Ownership Actual/diluted</th>
</tr>
</thead>
<tbody>
<tr>
<td>Common stock</td>
<td>Founders</td>
<td>100/100</td>
<td>$0</td>
<td>40%/40%</td>
</tr>
<tr>
<td>Preferred stock</td>
<td>Seed investors</td>
<td>100/100</td>
<td>$100</td>
<td>40%/40%</td>
</tr>
<tr>
<td>Preferred stock</td>
<td>Series A investors</td>
<td>100/100</td>
<td>$200</td>
<td>20%/20%</td>
</tr>
<tr>
<td>Options</td>
<td>Employees</td>
<td>20/20</td>
<td>$0</td>
<td>10%/10%</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td></td>
<td><strong>320/320</strong></td>
<td><strong>$300</strong></td>
<td><strong>100%/100%</strong></td>
</tr>
</tbody>
</table>

Finally, after the company is sold, the **capitalization table** is updated to reflect the exit. The company is sold for $1000, and the proceeds are distributed to the shareholders. The founders’ ownership is diluted to 30% of the company. The new shares are issued as **preferred stock**. The share numbers are recalculated to simplify the math.
<table>
<thead>
<tr>
<th>Asset class</th>
<th>Stakeholder</th>
<th>Shares</th>
<th>Cost</th>
<th>% Ownership</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Actual/diluted</td>
<td>Actual/diluted</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Common stock</td>
<td>Founders</td>
<td>30/30</td>
<td>$0</td>
<td>30%/30%</td>
</tr>
<tr>
<td>Common stock</td>
<td>Float</td>
<td>70/70</td>
<td>$100</td>
<td>30%/30%</td>
</tr>
<tr>
<td>Total</td>
<td>100/100</td>
<td>$100</td>
<td>100%</td>
<td>100%/100%</td>
</tr>
</tbody>
</table>

Table 2.6: A capitalization table after an exit

Float refers to the publicly held shares of a company. In this case, the float is 100 shares, and the founders’ ownership is 30% of the company.

2.2 JSON Schema

JSON Schema is a specification for JSON (JavaScript Object Notation) data, and provides an approach for defining the structure and constraints of JSON data [4]. JSON is a lightweight data interchange format, and is ubiquitously employed in web applications and APIs due to its simplicity and readability, and JSON Schema, allows developers to meticulously define the structure and constraints of JSON data. A comprehensive description of JSON Schema is provided in RFC8259 [5].

The versatility of JSON Schema is evident in its extensive range of validation rules and constraints. These include, but are not limited to, data types, required fields, minimum and maximum values, and regular expressions. Furthermore, JSON Schema supports custom validation rules and extensions, thereby offering developers the flexibility to define their own rules and constraints. This feature significantly enhances the adaptability of JSON Schema to cater to diverse and specific requirements.

JSON Schema is a powerful tool that leverages the simplicity and readability of JSON, making it human-friendly and easy to comprehend. It employs hypermedia principles, allowing schemas to reference other schemas through a Universal Resource Identifier (URI). This feature enhances the modularity and reusability of schemas, thereby promoting efficient schema design and management.

In the context of designing messages for HTTP application programming interfaces (APIs), JSON Schema is exceptionally adequate. It provides robust validation capabilities, ensuring the integrity and consistency of data communicated through APIs. Not only can it validate
the presence of all necessary arguments, but it also offers basic format validation. This means it can check if the data conforms to the specified types, patterns, or other constraints, thereby ensuring that the data received or sent via APIs adheres to the expected structure and format. This comprehensive validation capability significantly enhances the reliability and robustness of HTTP APIs, making JSON Schema an indispensable tool in modern web development.

2.2.1 Validating the presence of specific keys in a document

JSON Schemas can be used to check the presence of a set of valid keys of a document, as well as the set of valid values for that key. The values for each key are given a JSON Schema themselves. This means that JSON Schemas can compose with other JSON Schemas.

The structure and acceptable formats for the fields can be defined in terms of types, enumerations, and constants. Table 2.7 shows the available validations in JSON Schema.

<table>
<thead>
<tr>
<th>Validation Keyword</th>
<th>Specification</th>
</tr>
</thead>
<tbody>
<tr>
<td>type</td>
<td>of the data type (e.g., string, number, object, array, boolean, null)</td>
</tr>
<tr>
<td>enum</td>
<td>a predefined list of acceptable values</td>
</tr>
<tr>
<td>const</td>
<td>a constant value that the data must match</td>
</tr>
</tbody>
</table>

Table 2.7: Available Validations in JSON Schema (Types and enums)

2.2.2 Composing schemas to form more complex schemas

Schemas can be composed using composition keywords. Any schema might refer to other schemas by a URI, and the referred schema is expected to be found at that URI. The composition keywords are shown in table 2.8.
### Table 2.8: Composition Operators in JSON Schema

<table>
<thead>
<tr>
<th>Operator</th>
<th>Specification</th>
</tr>
</thead>
<tbody>
<tr>
<td>allOf</td>
<td>all the schemas must be valid for the document to be valid.</td>
</tr>
<tr>
<td>anyOf</td>
<td>at least one of the schemas must be valid for the document to be valid.</td>
</tr>
<tr>
<td>oneOf</td>
<td>exactly one of the schemas must be valid for the document to be valid.</td>
</tr>
<tr>
<td>not</td>
<td>the schema must not be valid for the document to be valid.</td>
</tr>
</tbody>
</table>

**2.2.3 Validating numeric, string and array values**

Numeric validation is restricted to the value of a single field, which can be constrained to be a multiple of a constant, or to be within a range. Table 2.9 shows the available numeric validations in JSON Schema.

<table>
<thead>
<tr>
<th>Validation Keyword</th>
<th>Specification</th>
</tr>
</thead>
<tbody>
<tr>
<td>multipleOf</td>
<td>that a numeric instance is divisible by this keyword’s value</td>
</tr>
<tr>
<td>maximum</td>
<td>the maximum numeric value</td>
</tr>
<tr>
<td>exclusiveMaximum</td>
<td>a numeric instance to be strictly less than this keyword’s value</td>
</tr>
<tr>
<td>minimum</td>
<td>the minimum numeric value</td>
</tr>
<tr>
<td>exclusiveMinimum</td>
<td>a numeric instance to be strictly greater than this keyword’s value</td>
</tr>
</tbody>
</table>

Table 2.9: Available Validations in JSON Schema (Numbers)

Strings can be validated by their length, or by a regular expression. Table 2.10 shows the available string validations in JSON Schema.
Table 2.10: Available Validations in JSON Schema (Strings)

<table>
<thead>
<tr>
<th>Validation Keyword</th>
<th>Specification</th>
</tr>
</thead>
<tbody>
<tr>
<td>maxLength</td>
<td>the maximum length of a string</td>
</tr>
<tr>
<td>minLength</td>
<td>the minimum length of a string</td>
</tr>
<tr>
<td>pattern</td>
<td>a regular expression that a string must match</td>
</tr>
</tbody>
</table>

Arrays can be validated by specifying a schema for its elements, by the number of elements, and by whether the elements must be unique. Table 2.11 shows the available array validations in JSON Schema.

Table 2.11: Available Validations in JSON Schema (Collections)

<table>
<thead>
<tr>
<th>Validation Keyword</th>
<th>Specification</th>
</tr>
</thead>
<tbody>
<tr>
<td>items</td>
<td>constraints for array items</td>
</tr>
<tr>
<td>maxItems</td>
<td>the maximum number of items in an array</td>
</tr>
<tr>
<td>minItems</td>
<td>the minimum number of items in an array</td>
</tr>
<tr>
<td>uniqueItems</td>
<td>that all items in an array must be unique</td>
</tr>
</tbody>
</table>

Object validation is the most basic of all, and is used to specify what keys are allowed and required in any object. Table 2.12 shows the available object validations in JSON Schema.

Table 2.12: Available Validations in JSON Schema (Objects)

<table>
<thead>
<tr>
<th>Validation Keyword</th>
<th>Specification</th>
</tr>
</thead>
<tbody>
<tr>
<td>properties</td>
<td>constraints for object properties</td>
</tr>
<tr>
<td>required</td>
<td>required properties in an object</td>
</tr>
<tr>
<td>additionalProperties</td>
<td>whether additional properties are allowed</td>
</tr>
</tbody>
</table>

2.2.4 Achievements and limitations of JSON Schema

JSON Schema is supported by many tools [6]: validators, schema generators and code generators. There are validators for all major languages, and the schema generators can
generate schemas from code, data, and models. Code generators can implement basic Web-based user interfaces and generate data based on schemas. It is also closely integrated with the OpenAPI specification, which is the de facto standard for describing HTTP APIs.

JSON Schema can validate data syntactically, which is adequate for a file or data exchange format. However, JSON Schema provides little in terms of expressiveness in sets, relations, and logic. It is not possible to express constraints such as “the sum of the shares of all stakeholders must be equal to the total number of shares issued by the issuer”. In JSON Schema, we cannot rule out cyclical sequences of security trades because the language lacks a closure operator. JSON Schema cannot reason about any relational properties of instances.

2.3 The Open Cap Table format

Our analysis is based on the following commit. The reader can run the command to fetch the specific files from GitHub.

```bash
git clone https://github.com/Open-Cap-Table-Coalition/Open-Cap-Format-OCF
git checkout 20f3ede62d1f5bdbef16ae1edfa98c34fbda2610
```

2.3.1 File format

The Open Cap Table format defines a package, or set, of JSON files, for storing data on transactions and business entities. Adopting the format means being able to export the capitalization table data according to the format.

The manifest contains metadata about the other files, which contain data. The data files either contain immutable entities which participate in transactions or the transactions themselves, which are the events that change the state of the entities. Table 2.13 shows the files in the format.
2.3.2 The existence of an implied conceptual model within the data model

Although the OCF specifies a set of files by defining its contents as JSON documents with associated JSON Schema, it is built on top of an implied conceptual model that underlies the data, which is just a representation.

This conceptual model implied in the data model is of great value for us. We will therefore present its organizing principles, describe its structure, and discuss the key components of the model while identifying the design patterns used to represent them.

2.3.3 Organizing principles of the underlying model

The format is given as a set of JSON Schemas. The schemas are used to validate the data files. Each schema is defined in a file ending with .schema.json.

The schemas are organized according to two principles, which are reflected in the directory structure of the repository:

- Technical building blocks: enums, types and objects
- Conceptual blocks: entities, transactions, conversion mechanisms, rights and triggers, and vestings

Technically, types (in OCF terminology) define structures (expected keys and associated...
validation) that are reused in primitive objects (in the sense JSON objects and documents) and in Enum (enumerations of constant values).

2.3.4 Opening example of the complete lifetime of a security

As we analyze the format, we will see how it can store a case such as the following:

1. Issuance Event: 1,000 shares of preferred stock are issued to Bob (generates a new security ID)
2. Acceptance Event: Bob accepts the shares (refers to the security ID generated in transaction 1)
3. Conversion Event: Bob converts 500 shares to common stock (refers to the security ID generated in transaction 1)

   Since this is a partial transaction, beyond issuing a security for 500 shares of common stock, we must issue new preferred stock for the remaining 500 shares as well

4. (a) Issuance Event: 500 shares of common stock are issued to Bob (a new security ID is generated)
   (b) Issuance Event: 500 shares of preferred stock issued to Bob (a new security ID is generated)
5. Transfer Event: Bob transfers 500 shares of preferred stock to Alice

   Again, this is a partial transaction, so we must issue new preferred stock for the remaining 500 shares as well

6. (a) Issuance Event: 500 shares of preferred stock are issued to Frank (a new security ID is generated)
   (b) Issuance Event: 500 shares of preferred stock issued to Bob (a new security ID is generated)

7. Repurchase Event: Issuer repurchases 500 shares of common stock from Bob (This is a complete transaction, so no new security ID is generated)
Conceptually, the model defines entities which are referenced to in transactions. Entities include issuers, stakeholders, stock classes, stock option plans, vesting terms and stock legend templates. Transactions include security trades, conversions, grants, exercises, cancellations, terminations, and vesting events. Conversion mechanisms, rights, and triggers are used to define the conversion of securities. Vestings are used to define vesting terms.

By considering all transactions, we can derive the state of the entities at any point in time.

### 2.3.5 A note on the current folder structure of the OCF

The two principles are mixed in the original OCF distribution, regarding the folder structure. Nevertheless, the principles are useful enough for our purposes. We will use them to describe the OCF schemas. Figure 2.2 shows the directory structure of the OCF distribution.

```plaintext
\texttt{schema/}
  \texttt{  _schema/enums/}
  \texttt{  _schema/files/}
  \texttt{  _schema/objects/}
    \texttt{    _schema/transactions/}
  \texttt{  _schema/primitives/}
    \texttt{    _schema/files/}
    \texttt{    _schema/objects/}
    \texttt{    _schema/types/}
  \texttt{  _schema/types/}
    \texttt{    _schema/conversion_mechanisms/}
    \texttt{    _schema/conversion_rights/}
    \texttt{    _schema/conversion_triggers/}
    \texttt{    _schema/vesting/}
```

Figure 2.2: Directory structure of the OCF distribution.
2.3.6 General use types

There are 50 available types in the OCF, of which 22 are types for general use, such as address types, currencies, monetary values et cetera. Figure 2.3 details the directory structure of the general use types.

```
schema/
  types/
    Address.schema.json
    CapitalizationDefinition.schema.json
    ContactInfo.schema.json
    CountryCode.schema.json
    CountrySubdivisionCode.schema.json
    CurrencyCode.schema.json
    Date.schema.json
    Email.schema.json
    File.schema.json
    InterestRate.schema.json
    Md5.schema.json
    Monetary.schema.json
    Name.schema.json
    Numeric.schema.json
    Percentage.schema.json
    Phone.schema.json
    Ratio.schema.json
    SecurityExemption.schema.json
    ShareNumberRange.schema.json
    StockParent.schema.json
    TaxID.schema.json
    TerminationWindow.schema.json
```

Figure 2.3: Types in the OCF for general use
2.3.7 Types supporting transactions and securities

Beyond the general use types for common concepts such as addresses and monetary values, there are types defining the entities and transactions that participate in the lifetime of a capitalization table. Figure 2.4 shows the directory structure of the types supporting transactions and securities.

```
schema/
  types/
    conversion_mechanisms/
      CustomConversionMechanism.schema.json
      SAFEConversionMechanism.schema.json
      NoteConversionMechanism.schema.json
      PercentCapitalizationConversionMechanism.schema.json
      FixedAmountConversionMechanism.schema.json
      RatioConversionMechanism.schema.json
```

Figure 2.4: Conversion mechanisms

Figure 2.5 shows the directory structure of the conversion rights.

```
schema/
  types/
    conversion_rights/
      WarrantConversionRight.schema.json
      ConvertibleConversionRight.schema.json
      StockClassConversionRight.schema.json
```

Figure 2.5: Conversion rights

Figure 2.6 shows the directory structure of the vesting types.
Figure 2.6: Vesting events and conditions, periods and portions

Figure 2.7 shows the directory structure of the conversion triggers.

Figure 2.7: Conversion triggers, divided in elective and automatic

2.3.8 Enums

 Enums are constant values. Figure 2.8 shows the directory structure of the enums.
They at principle look simple or trivial; actually, they consolidate important domain expertise. The Allocation type, for example, defines that shares in a vesting term can be allocated with different rounding methods, and whether to allocate shares unevenly divisible by the number of vesting periods by front-loading (i.e., allocating the remainder to the first vesting period) or back-loading (i.e., allocating the remainder to the last vesting periods). From the author’s experience, this sort of low-level business rule detail takes a large part of developing systems for financial domains. There are specific enums for period-
icity classes, different methods of calculating the day of vesting in the month (i.e., business
days versus calendar days), and different methods of calculating interest accruals, rounding
methods et cetera. These details are often overlooked during the design of a system for
managing financial information, so we value that they are explicitly defined in the OCF.

2.3.9 Transactions

Transactions are the actions performed on securities. Transactions are organized in
primitive transactions and specific transactions. Primitive transactions represent the
different kinds of transactions in the financial domain. Figure 2.9 shows the directory
structure of the primitive transactions.

```
schema/
  __primitives/
   __objects/
    __transactions/
      __acceptance/Acceptance.schema.json
      __cancellation/Cancellation.schema.json
      __conversion/Conversion.schema.json
      __exercise/Exercise.schema.json
      __issuance/Issuance.schema.json
      __reissuance/Reissuance.schema.json
      __release/Release.schema.json
      __repurchase/Repurchase.schema.json
      __retraction/Retraction.schema.json
      __SecurityTransaction.schema.json
      __StockClassTransaction.schema.json
      __StockPlanTransaction.schema.json
      __Transaction.schema.json
      __transfer/Transfer.schema.json
```

Figure 2.9: Primitive transactions in the OCF

The primitive transactions, with additional fields, are specialized to each of four different
types of securities: stocks, convertible notes, warrants, and plan securities. We group
them by the asset class they specialize in. Figure 2.10 shows the directory structure of
the specialized transactions.
Figure 2.10: Transactions relevant to stock options and vesting

Figure 2.11 shows the directory structure of the specialized transactions for stocks.

Figure 2.12 shows the directory structure of the specialized transactions for warrants.
Figure 2.12: Transactions relevant to warrants

Figure 2.13 shows the directory structure of the specialized transactions for convertibles.

Figure 2.13: Transactions relevant to convertibles

Additional transactions adjust parameters of stock classes and stock option plans. Figure 2.14 shows the directory structure of the adjustment transactions.

Figure 2.14: Adjustment transactions in the OCF

37
2.3.10 Entities and other objects

Beyond transactions being entities, the OCF defines entities for each one of the files in the manifest (cf. page 28).

2.4 Key components and patterns in the OCF

The OCF has three key logical components: tracing of transactions, rules for vesting, and rules for convertible securities, as shown in table 2.14.

<table>
<thead>
<tr>
<th>Transaction &amp; tracing</th>
<th>Transactions that are linked by security identifiers (i.e., the issuance and cancellation of a security refer to a common security identifier)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Vesting</td>
<td>Composable rules for both schedule-based and event-based vesting</td>
</tr>
<tr>
<td>Convertible securities</td>
<td>Composable rules for converting securities, typically applied in the case of debt that can be converted to stock shares</td>
</tr>
</tbody>
</table>

Table 2.14: Key components in the OCF

2.4.1 Transaction tracing system

Since a capitalization table is a snapshot after a set of transactions have been accumulated, one component of the OCF is the support of traceable transactions. The general principle is that transactions are objects which are stored in the transactions file (cf. page 28), which refer to securities. A security is a financial asset that can be bought and sold. Stocks, options, and debt notes are all securities.

Securities do not have a specific schema in the OCF—they are given implicitly as correlated identifiers.

The key ideas behind the transaction tracing systems are:

- Securities have a initial and a terminal transaction.
- Issuances (including re-issuances) and exercises are initial transactions.
- Cancellations, retractions, repurchases are terminal transactions.
• Transfers are both initial and terminal transactions, as they extinguish the initial security and create a new security (the result security).

• Partial cancellations are possible by extinguishing the original security and generating a new security with the remaining balance.

• Partial transfers are possible by extinguishing the original security and generating a new security with the remaining balance.

To compute the state of a security, all transactions that refer to it must be traced. The state of a security is the sum of all issuances minus the sum of all cancellations, retractions, repurchases, and transfers. This chaining is what enables auditability and traceability in the system.

Pictorially, the scheme is as follows:

**An issuance**

Figure 2.15 shows an issuance as an input-output system (in a special case with no inputs).

![An issuance](image)

A issuance requires no security as input, and always produces a resulting security.

**A partial cancellation**

Figure 2.16 shows a partial cancellation as an input-output system. The existing security, of which some number of shares will be cancelled, is the input. The output is the security with the balancing shares.
Cancelling 50 of 100 shares implies a partial cancellation, and a balance security is thus generated.

**A partial transfer**

Figure 2.17 shows a partial transfer as an input-output system. The existing security, of which some number of shares will be transferred, is the input. The outputs are the new security and the balancing shares.

Transferring 50 of 100 shares implies a partial transfer, and a balance security is thus generated.

**Example — the transfer schema**

The full listing for the transaction schemas can be found in the appendix, but by giving the transfer schema, we can depict the general pattern of the transaction schemas.
Listing 1 shows the primitive transfer schema, which must be extended by the specific transfer schemas for each **asset class**. It states that every transaction must present a `security_id` field.

```json
{
    "$schema": "http://json-schema.org/draft-07/schema",
    "$id": "$/primitives/objects/transactions/SecurityTransaction.schema.json",
    "title": "Primitive - Security Transaction",
    "description": "Abstract transaction object to be extended by all transaction objects that deal with individual securities",
    "type": "object",
    "properties": {
        "security_id": {
            "type": "string"
        }
    },
    "required": ["security_id"],
}
```

Listing 1: Primitive transaction schema

In listing 2, we see the transfer schema, which extends the primitive transaction schema.
In the listing above, we can clearly see the pattern of an input security being consumed and a new security being created. The balance_security_id field is used to indicate that a partial transfer is being made, and the resulting_security_ids field is used to indicate that multiple securities are being created.

Transactions that affect different asset classes (e.g. common stock, preferred stock, options, warrants, etc.) have different behavior: options have transactions specific to vesting and exercises and objects to represent the vesting schedule, warrants have transactions to represent the vesting schedule, and convertible securities have also objects representing the conversion mechanisms and parameters.

The full cross-referenced table of asset classes and their transaction types is given below. For clarity, we omit concepts in the OCF that are transactions with no affected security, such as stock class splits and adjustments and stock option pool increases. Table 2.15

Listing 2: Transfer schema in the OCF

```json
{
  "$schema": "http://json-schema.org/draft-07/schema",
  "$id": "$/primitives/objects/transactions/transfer/Transfer.schema.json",
  "type": "object",
  "properties": {
    "consideration_text": {
      "type": "string"
    },
    "balance_security_id": {
      "type": "string"
    },
    "resulting_security_ids": {
      "title": "Security Transfer - Resulting Security ID Array",
      "type": "array",
      "items": {
        "type": "string"
      },
      "minItems": 1,
      "uniqueItems": true
    }
  },
  "required": ["resulting_security_ids"]
}
```
<table>
<thead>
<tr>
<th>Transaction Type</th>
<th>Convertible</th>
<th>Options</th>
<th>Stock</th>
<th>Warrant</th>
</tr>
</thead>
<tbody>
<tr>
<td>Acceptance</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Cancellation</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Conversion</td>
<td>Yes</td>
<td></td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>Exercise</td>
<td></td>
<td>Yes</td>
<td></td>
<td>Yes</td>
</tr>
<tr>
<td>Issuance</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Reissuance</td>
<td></td>
<td></td>
<td></td>
<td>Yes</td>
</tr>
<tr>
<td>Release</td>
<td></td>
<td></td>
<td></td>
<td>Yes</td>
</tr>
<tr>
<td>Repurchase</td>
<td></td>
<td></td>
<td></td>
<td>Yes</td>
</tr>
<tr>
<td>Retraction</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Transfer</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
</tbody>
</table>

Table 2.15: Transactions and their supported security types

shows the transactions and their supported security types.
2.4.2 Vesting system

As we discussed above in 2, stock options follow a vesting system whereby options are released to optionees over time.

Figure 2.18 shows the directory structure of the vesting schemas.

```
schema/
  |   enums/
  |     VestingDayOfMonth.schema.json
  |     VestingTriggerType.schema.json
  |   files/
  |     VestingTermsFile.schema.json
  |   objects/
  |     transactions/vesting/
  |       VestingAcceleration.schema.json
  |       VestingEvent.schema.json
  |       VestingStart.schema.json
  |       VestingTerms.schema.json
  |   primitives/
  |     types/vesting/
  |       VestingConditionTrigger.schema.json
  |       VestingPeriod.schema.json
  |     vesting/
  |       VestingCondition.schema.json
  |       VestingConditionPortion.schema.json
  |       VestingEventTrigger.schema.json
  |       VestingPeriodInDays.schema.json
  |       VestingPeriodInMonths.schema.json
  |       VestingScheduleAbsoluteTrigger.schema.json
  |       VestingScheduleRelativeTrigger.schema.json
  |       VestingStartTrigger.schema.json
```

Figure 2.18: Vesting schemas in the OCF

The key object is the Vesting Term object. This object is used to represent the vesting terms of a security (vesting term is a synonym for a schedule). The responsibility of the vesting term is factored into more schemas. The vesting term itself defines an allocation type and points to an array of vesting conditions. Listing 3 shows the vesting term schema.
The Vesting Condition, in turn, defines a number of shares and points to an array of vesting triggers. Figure 4 shows the vesting condition schema.
Each vesting trigger, in turn, is one of four kinds:

- **Vesting Start** triggers after the **Cliff period** has passed
- **Vesting Schedule Absolute** triggers on a specific date
- **Vesting Schedule Relative** triggers after a specific period of time has passed relative to another date
- **Vesting Event** triggers after a specific event has occurred

In the most typical case, that of one year **Cliff period** and monthly vesting over the next three years, the corresponding vesting term would contain $1 + 36$ vesting conditions, the first of which is triggered by a vesting start event, and the remaining 36 of which are
triggered by a vesting schedule relative event (regarding the vesting start), with a period of one month.

By composing the triggers described above, one can define a variety of vesting terms. This composition of triggers is nothing short of a proto-domain specific language. That is why the Open Cap Table Coalition uses the term “vesting graph” for the resulting composition of triggers and conditions (see [7]).

The vesting term system can accommodate the most basic vesting scenarios—we will improve upon, so it can support a richer set of vesting terms, by expanding the idea of a domain-specific language.

### 2.4.3 Convertible securities

In this section, we show how conversion mechanisms model transformation of securities from one type to another, which usually involves either the conversion of preferred stock into common or the conversion of debt into gsequity. Figure 2.19 shows the directory structure of the convertible securities system.

The building blocks of conversions are mechanisms, rights, and triggers. There are mechanisms for converting to a given percentage of a company’s capitalization, as a fixed number of shares, at a given ratio, and in the case of convertible debt notes, the mechanism is given in terms of a valuation cap and a discount, and also gives concepts related to interest payments.

A conversion right complements the conversion mechanism by giving the stock class of the resulting security. Conversion triggers are more interesting, and specify whether the conversion is triggered electively, at a given date or event, or whether it is automatic.

### 2.5 Discussion

#### 2.5.1 Advantages and achievements of the OCF

We see OCF as incredibly valuable given how much knowledge it gathers in a single specification. The choice of design patterns results in a data model that can fulfill the requirements of auditability (in special regarding transaction tracing) and flexibility (via the domain-specific languages implicit in the vesting and conversion systems).

After reviewing the Open Cap Table Format, we can agree that:
Figure 2.19: The convertible securities system
1. The OCF consolidates a wealth of expert knowledge in the capitalization tables domain.

2. The data format was explicitly designed to be auditable: ultimately there is a set of immutable entities that when replayed together with the transactions give the current state of the capitalization table.

3. The vesting system and the conversion system, in particular, are based on a complex tangle of business rules which are modeled in the OCF (such is the case that the corresponding part of the model looks like a domain-specific language).

4. Many useful auxiliary types are provided to model dates, addresses, monetary values, which are useful in itself.

2.5.2 Disadvantages and limitations of the OCF

The choice of JSON Schema can arguably be defended because it is widespread, easy to use and can validate data, by its purpose.

However, we believe that the use of JSON Schema is a limitation to the OCF, because of the characteristics of JSON Schema’s semantics.

JSON Schema fails to express invariants of the system. The JSON Schema itself cannot distinguish wrong transaction traces from correct ones. In fact, the samples given in the repository are given only syntactically correct. The documents in the samples are well-formed, so to say. But they are not semantically correct. The examples are trivial.

In order for the OCF to reach its full potential as a standard, correct validators must be available. In this regard, we must note that the choice of JSON Schema carries a number of downsides.

For any language to be able to validate the OCF data, it must be capable of expressing some facts that are not at all possible to validate with JSON Schema:

1. To validate that no more options have been exercised than those granted, a language must be able to reason about the fields of associated entities, such a stock plan and its plan securities, and perform sums. JSON Schema has no way to apply this validation. Accounting equations cannot be verified in any practical way with JSON Schema.

2. There is nothing that JSON Schema can do to make sure that there are no cycles as we trace securities. Transitive relations must be considered when tracing securities.
Critically, JSON Schema is incapable of validating the OCF properly.

Secondly, a format also must be able to express a wider range of business logic: although the support rules for vesting are expressive enough for a broad range of cases, it can only express a conjunctive condition (all triggers must be satisfied) and it can only refer to points or intervals going forward and unbounded in the future. But this precludes the combination of a vesting condition that triggers when a milestone is met within a deadline because we would have to model an “until” operator.

We can improve the model by making the vesting term more expressive, and more capable of modeling the conditions that can be found in legal documents in the wild.

We can improve on how easy it is to validate and improve the expressiveness of the original OCF by adopting a stronger modeling framework: Alloy and the Alloy Analyzer. This should keep the business knowledge that is available in the OCF, while making it more rigorous.
Part II

Models
Chapter 3

A new model for capitalization tables

This chapter delineates a model for a system tasked with managing capitalization tables, centered around the pivotal concepts of securities and transactions.

The proposed model serves as an initial framework for the development of validation tools and information management applications. Additionally, it lays the groundwork for code-normative documents, such as smart contracts.

First, let us quickly review the Alloy modeling language, and review related works.

3.1 An introduction to Alloy

Alloy, first introduced by Daniel Jackson in 2002 [8], is today a complete framework for modeling and analyzing software systems. We say it is a framework because it is composed of more than just a modeling language.

The idea underlying Alloy is that “software should be build on abstractions”, and that by picking the right ones, programming should “naturally from design”.

Alloy is distributed as the Alloy Analyzer, which features:

- A modeling language
- A visualizer for metamodels and model instances
- A bounded model checker
- A standard library for common data structures (sequences, graphs, relations, etc.)

By combining the three features above, an Alloy user can quickly iterate on a model, visualize and explore instances, and express and check properties. An Alloy model can focus both on the concepts underneath the system being modeled, on the data model and relationships and on more operational aspects of the system by modeling relations between two or more states.

### 3.1.1 Parts of an Alloy model

Every alloy model is composed of signatures, predicates, facts, functions, assertions and commands:

1. Signatures: the basic building blocks of Alloy models, they represent entities in the domain. Signatures have fields that relate them to other signatures.

2. Functions: A function in Alloy relates instances of signatures to relations. An alloy function is a function like in any programming language.

3. Predicates: A predicate in Alloy is a function that returns a boolean value. Predicates are used to express properties of the model.

4. Facts: A fact is a predicate that must always be true in the model. A fact actively constrains the model by discarding instances that do not satisfy it.

5. Assertions: An assertion is predicate that can be given to a check command, and expresses our expectations of the model. Assertions do not constrain the model effectively.

6. Commands (Checks and runs): The run finds examples of the model that satisfy all constraints, while the check command finds counter-examples that violate a given assertion.

A full presentation of Alloy is beyond the scope of this work, but can be found in the Software Abstractions book by Daniel Jackson [9].
3.2 A brief overview of Alloy related literature

Alloy is cited in more than 1,200 papers, with many extensions and applications in software design and modeling. Alloy∗ is a higher-order extension of Alloy that can be used for program synthesis, since it supports higher-order quantification [10]. aRby is an embedding of Alloy in Ruby [11].

A number of other modeling languages have been translated to or partially modeled in Alloy[12], including UML, i∗, CVL, Event-B and, Z, OCL. Alloy can also be used in conjunction with verification languages such as ACL2 or Isabelle.

Alloy has been used in a wide range of applications in software engineering, database design, security analysis [13][14], multiagent negotiations [15]. It has also been applied to modeling beyond computer science such as a model for central bank policy [16]. A model of the same-origin-policy used in web browsers can be found in the 500 Lines or Less open-source book [17].

We were not able to find previous models of capitalization tables in Alloy.

Of paramount importance is the model’s capability to fully validate and calculate the present state of the system without necessitating the replay of all transactions. This constitutes a substantial advancement over the original model. Conversely, the Open Cap Table Coalition’s framework lacks sufficient specification to compute the current state of a capitalization table, which is a critical prerequisite for the development of optimal tooling.

Alloy played an instrumental role in iterating through numerous design considerations, inclusive of those found in the Open Cap Table Coalition, culminating in the formulation of the present model.

3.3 Motivation

Should a model within the same domain as the Open Cap Table Coalition (OCF) and incorporating many of its insights exist in a more expressive language, it could facilitate the specification and implementation of tools for the validation, management, and accurate reporting of capitalization tables.

The current limitation stems from the OCF’s choice of JSON Schema as its modeling language, constraining the application of its extensive industry knowledge to rudimentary
data validation tasks.

Such specialized tooling would address a highly resource-intensive, real-world issue, consuming substantial human capital, as elaborated in Chapter 1.

### 3.4 Relation to the Open Cap Table Format

The Open Cap Table Format (OCF) is limited to elementary data validation and lacks the expressive power to codify community-accepted business rules. While it can conduct basic field-value validation, it is unable to enforce structural or arithmetic rules, thereby constraining its applicability. Such limitations are inherent to its chosen modeling language, JSON Schema.

In contrast, our enhanced model capitalizes on the Alloy language’s capacity for formalizing both constraints and update rules specific to a capitalization table. The superior expressiveness of Alloy relative to JSON Schema leads to a concomitantly enhanced range of capabilities. Table 3.1 delineates a comparison between the two models, evidencing our contributions.

<table>
<thead>
<tr>
<th>Our Model</th>
<th>Original Model</th>
</tr>
</thead>
<tbody>
<tr>
<td>Explicit representation of current state</td>
<td>Necessitates transaction replay for state</td>
</tr>
<tr>
<td>Structural Constraints</td>
<td>Field-Value Validation</td>
</tr>
<tr>
<td>Accounting Constraints</td>
<td>Rudimentary Integer Range Constraints</td>
</tr>
<tr>
<td>Advanced Vesting System with AND/OR/NOT</td>
<td>Basic Vesting System with OR</td>
</tr>
</tbody>
</table>

Table 3.1: Comparison between our model and the original model

### 3.5 Methodology

Our methodology for model construction adopts an iterative approach, consistent with best practices for employing Alloy:

1. Initially, we isolate each discrete concept within the Open Cap Table Coalition’s original model. Corresponding Alloy signatures are created for these concepts, and anticipated invariant properties are defined.

2. Subsequent to establishing initial properties and signatures, we assess their validity.
Should any property prove inconsistent, we amend the relevant signature with additional constraints to rectify the discrepancy.

3. Instance model visualizations are rigorously examined to confirm alignment with our theoretical expectations and to identify any conspicuous inconsistencies.

4. This iterative process continues until the model fulfills our pre-established conditions and criteria.

Several factors contribute to the efficacy of our methodology. First, Alloy enables swift feedback loops, thereby streamlining the iterative development cycle. Second, the amalgamation of automated verifications and manual scrutiny serves as a robust validation mechanism. Additionally, Alloy’s capacity to transform problems into boolean satisfiability issues and identify unsatisfiable subsets (UNSAT cores, in the literature) within the original constraints is invaluable for debugging and for isolating root causes when the model diverges from expected behavior.

3.6 Key Concepts in the Model

Like its Open Cap Table Coalition predecessor, our Alloy model centers on securities and transactions. However, transactions in our model are state-transformative; they invalidate previous securities and instantiate new or balanced ones. This ensures an always-current, consistent state, encapsulated by structural and accounting constraints.

Our state-management approach diverges significantly from the original model. Transactions result in the deprecation of input securities and generation of output and balance securities, thus enabling local state validation. This removes the need for historical transaction replay and streamlines implementation, as the validity of each transaction becomes locally deterministic.

We focus on stock and plan securities, omitting less common types like warrants and convertible debts. Our model, however, remains extensible for future inclusion of these types, owing to the local nature of the constraints. This will become clear as we examine the specific signatures in Section 3.7.

3.6.1 Treatment of Securities

In the Alloy-based model, securities are managed according to the following computational principles:
1. Each security is anchored to an *initial transaction*, constituting its instantiation within the system. Optionally, a *terminal transaction* may demarcate its termination.

2. Securities are flagged as either *current* or *expired*.

3. Transition to an expired state occurs solely upon association with a *terminal transaction*, thus invalidating the security for subsequent transactions.

4. A security is instantiated either de novo or derived from a *parent security*, thereby inheriting attributes such as type and constraints.

5. Share quantities must remain positive, enforcing accounting integrity and regulatory compliance.

### 3.6.2 Treatment of transactions

In the formal Alloy-based model, transactions adhere to the following computational rules:

1. Transactions are composed of input, output, and balance securities. Input securities undergo transformation, output securities are instantiated, and balance securities capture residual shares not transferred or voided.

2. Transactions may function as either the *initial transaction* or *terminal transaction* for securities. The former activates a security, while the latter marks its termination.

3. Transactions may possess a *parent transaction*, which is accountable for generating the input securities. Attributes like transaction type and constraints can be inherited from this parent transaction.

These rules induce a directed graph, with parent-child relationships between securities and transactions forming its edges. By representing these edges as a partial ordering relation, cycle detection and prevention are facilitated. Cycles introduce logical contradictions, such as securities self-referencing as parents. Leveraging Alloy’s capacity for transitive closure analysis, cycles can be automatically identified and invalidated. By conforming to these rules and utilizing Alloy’s analytical capabilities, we uphold the integrity of share accounting, disallowing both negative and zero quantities.
3.7 Signatures for Securities, Transactions, and Stakeholders

In the model, signatures serve as the foundational entities. Specifically, we define signatures for three main categories: securities, transactions, and stakeholders. Within these, we further classify different types of securities and transactions through additional signatures.

The signatures for both securities and transactions are designed as abstract. Subsequently, the signatures for their respective types are extensions of these abstract foundations. This structure adheres to a prevalent pattern in Alloy modeling.

Beyond securities and transactions, there are signatures for the vesting system, constraints, and checks (properties to be verified under a bound scope size).

3.7.1 Abstract security signature

All transactions and the two kinds of securities extend an abstract signature. The signature for an abstract security shown in listing 5. It contains the relationships with transactions (initial and terminal), and the parent security (which might be none, in the case of issuances). A pictorial representation of the signature is shown in figure 3.1.

Figure 3.1: Metamodel of the abstract security
The Security abstract signature is extended by the StockTx and PlanTx signatures. Listings for each signature are shown in listings 6 and 7. Structural constraints are defined in the transaction signatures, while accounting constraints are defined on both.

A security must always represent some shares; instances with zero shares are not allowed. Also, an implication from the model. Consider that a partial cancellation or transfer may not result in zero shares, since in that case it is a total cancellation or transfer, which create no new securities, and thus no securities with zero shares.

We enforce that the sum of granted, vested and exercised shares is always positive, and that the number of vested shares is always less than or equal to the number of granted shares, which is always less than or equal to the number of exercised shares, that is, \(0 \leq exercisedShares \leq vestedShares \leq grantedShares\).
Listing 7: Signature for a plan security

3.7.2 Abstract signature for transactions

The signature for an abstract transaction is similar. It has relationships to input, output and balance securities and to its parent transaction. Since transactions might have or might have not input, output and balance securities, we use the lone keyword to indicate that the relationship is optional. The signature is shown in listing 8. A pictorial representation of the signature is shown in figure 3.2.
Figure 3.2: Metamodel of the transaction signature, including relationships between transactions and securities, and the extensions PlanTx and StockTx

Listing 8: Abstract signature for a transaction

3.7.3 Concrete transactions signatures

We will consider nine different transactions in our model, sufficient for both stock and plan securities. The transactions are listed in table 3.2. We show which transactions have or have not input, output and balance securities. This shows how every transaction follows a general form. Issuance transactions only produce new securities, while cancellation transfer consume the original security and if partial might generate a balance security. Vesting transactions will generate a new security with updated quantities of granted, vested or
exercised shares. Transfers always create a new security for the new stakeholder, and if partial might generate a balance security. Exercises will be an interesting transaction, since they are the only transaction that consume a plan security but output a stock security.

Table 3.2: Transactions in the model

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Stock Issuance</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>Plan Issuance</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>Stock Total Cancellation</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>Stock Total Transfer</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>Plan Vesting Event</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>Plan Vesting Date</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>Stock Partial Transfer</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Stock Partial Cancellation</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Plan Exercise</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
</tbody>
</table>

The pictorial representation of all types of stock security transactions and how they are related via the extension relation is shown in figure 3.3. The equivalent representation for plan security transactions is shown in figure 3.4. Note that the only direct relationship between transactions is the parent relationship.
From the perspective of securities, transactions are associated through the `initialTx` and `terminalTx` relations. Conversely, from the standpoint of transactions, the `inputSecurity`, `outputSecurity`, and `balanceSecurity` relations establish connections to securities.

The `parentSec` and `parentTx` relations induce graphs over the domains of securities and transactions, respectively. Furthermore, the `initialTx` and `terminalTx` relations serve as the linking mechanisms between the graph of securities and the graph of transactions. The resultant bipartite structure is depicted in Figure 3.5.
Now we will detail the specific signatures for each transaction type, and provide examples of instances where those transactions are used.

**The Stock Security Issuance Transaction**

Stock securities are instantiated by Stock Issuance Transactions, which inject a specified number of shares, denoted as `issuedShares`, into the system. These transactions allocate the newly created security to a designated stakeholder, referred to as `issuedTo`. Additionally, stock securities can be introduced through plan security exercises, as elaborated in Section 3.7.3.

The constraints delineated in the signature specify that a Stock Issuance Transaction is
associated with a singular output security, which is of the stock type. This stock security
designates the issuance transaction as its initial transaction. The subsequent constraint
block stipulates that no other securities are linked to this particular transaction. The
third constraint block maps the attributes of the security to the corresponding attributes
in the transaction. Given that this operation pertains to an issuance, no input security
or parent transaction is necessitated. Lastly, the accounting constraint is straightforward,
manding that the number of shares be positive. Listing 9 shows the source code for the
Alloy signature.

Figure 3.6: An example instance of the stock issuance transaction
The Stock Security Partial Cancellation Transaction

In Stock Partial Cancellation Transactions, represented as `StockPartialCancellationTx`, the model extends `StockTx` by introducing `cancelledShares` to indicate the shares intended for cancellation. The constraints in the signature articulate that each transaction is linked to exactly one `balanceSecurity` and `inputSecurity`, while disallowing any `outputSecurity`. Additionally, both `inputSecurity` and `balanceSecurity` must be of the `StockSecurity` type. The model sets `balanceSecurity` as parented to `inputSecurity`, and uniquely ties their `initialTx` and `terminalTx` attributes to this transaction. Figure 3.7 presents an instance of this transaction.

Accounting constraints further refine the model’s rigor, and are a bit more involved than in the Stock Issuance Transaction. Specifically, `cancelledShares` must be positive and less than the total shares in `inputSecurity`. The resulting `balanceSecurity` shares are calculated as the difference between the shares in `inputSecurity` and `cancelledShares`, and this value also needs to be positive. Listing 10 shows the source code for this signature.
Figure 3.7: An example instance of the stock partial cancellation transaction

```plaintext
sig StockPartialCancellationTx extends StockTx {
  cancelledShares : Int
}

one balanceSecurity \&& one inputSecurity \&& no outputSecurity
balanceSecurity.parentSec = inputSecurity

balanceSecurity.initialTx = this
inputSecurity.terminalTx = this

no s : Security - balanceSecurity | s.initialTx = this
no s : Security - inputSecurity | s.terminalTx = this

balanceSecurity.owner = inputSecurity.owner

parentTx = inputSecurity.initialTx

pos[cancelledShares]
lt[cancelledShares, inputSecurity.shares]
leq[balanceSecurity.shares, sub[inputSecurity.shares, cancelledShares]]
pos[balanceSecurity.shares]
}
```

Listing 10: The stock partial cancellation transaction signature

The Stock Security Partial Transfer Transaction

In the Stock Partial Transfer Transactions, represented as StockPartialTransferTx, the model extends the foundational StockTx to include transferredTo, denoting the receiving stakeholder, and transferredShares, specifying the quantity of shares transferred. According to the signature constraints, each transaction is associated with exactly one balanceSecurity, inputSecurity, and outputSecurity, all of which must be of the StockSecurity type. These securities are interconnected in such a way that outputSecurity and balanceSecurity are both parented to inputSecurity, and...
their initialTx attributes are set to this transaction. Meanwhile, the terminalTx of inputSecurity is also specified as this transaction. Figure 3.8 shows an instance of a stock partial transfer.

The model imposes additional accounting and ownership constraints for internal consistency. Specifically, transferredShares must be a positive integer and less than the total shares in inputSecurity. The securities resulting from the transfer—the outputSecurity and balanceSecurity—are determined based on transferredShares and the remaining shares in inputSecurity, respectively. Both resulting securities must have a positive number of shares. Ownership rules are also enforced: outputSecurity is owned by the transferredTo stakeholder, while balanceSecurity retains the same owner as inputSecurity. The source code is available in Listing 11.

Figure 3.8: An example instance of the stock partial transfer transaction
The Stock Security Total Cancellation Transaction

In Stock Total Cancellation Transactions, encapsulated as `StockTotalCancellationTx`, the model dictates that each transaction is exclusively associated with a single `inputSecurity` of the `StockSecurity` type. The schema expressly prohibits the inclusion of either `outputSecurity` or `balanceSecurity`, and assigns this transaction as the `terminalTx` for the `inputSecurity` involved.
Figure 3.9 illustrates a representative instance of this transaction, and Listing 12 provides the corresponding source code. As depicted, the complexity of this particular transaction is markedly reduced compared to other transaction types.

Figure 3.9: An example instance of the stock total cancellation transaction

```
sig StockTotalCancellationTx extends StockTx {
    
    one inputSecurity && no outputSecurity && no balanceSecurity
    inputSecurity in StockSecurity
    inputSecurity.terminalTx = this

    no s : Security | s.initialTx = this
    all s : Security | s.terminalTx = this implies s = inputSecurity

    parentTx = inputSecurity.initialTx
}
```

Listing 12: The stock total cancellation transaction signature

The Stock Security Total Transfer Transaction

In Stock Total Transfer Transactions, denoted as StockTotalTransferTx, the model extends StockTx by introducing transferredTo to specify the receiving stakeholder. According to the constraints, each transaction uniquely links to one inputSecurity and one outputSecurity, while deliberately excluding any balanceSecurity. Both inputSecurity and outputSecurity are required to be of the StockSecurity type. The outputSecurity is parented to the inputSecurity and has its initialTx attribute set to this transaction, while the inputSecurity’s terminalTx is also defined as this transaction.

Figure 3.10 presents an instance of the transaction, and Listing 13 presents the source code.

Figure 3.10: An example instance of the stock total transfer transaction
Listing 13: The stock total transfer transaction signature

The Plan Security Issuance Transaction

The Plan Security Issuance transaction exhibits greater complexity compared to standard stock security transactions due to its adherence to a vesting system. As elaborated in Chapter 2, the vesting system establishes temporal regulations for the gradual release of securities, commonly observed in employee stock option grants.

The transaction necessitates a verification step to ensure that the number of shares specified in the conditions (refer to Section 3.8) aligns with the number of shares granted. Given that shares are subject to vesting and vesting is a prerequisite for exercises, the count of vested and exercised shares initiates at zero.

In the Alloy model, PlanIssuanceTx extends PlanTx and specifies additional fields such as issuedTo, grantedShares, and conditions. The constraints delineate that there is no inputSecurity, exactly one outputSecurity, and no balanceSecurity. Moreover, outputSecurity must belong to the category of PlanSecurity and should not have a parentSec. Other constraints ensure the consistency and integrity of the transaction, such
as the initialization of `vestedShares` and `exercisedShares` to zero and the requisite equality between `vestedShares` and the sum of `conditionedShares` specified in the `conditions`.

Figure 3.11 shows an example instance of a plan security issuance transaction, and Listing 14 presents the source code.

**Figure 3.11: An example instance of the plan issuance transaction**

```plaintext
sig PlanIssuanceTx extends PlanTx {
    issuedTo : one Stakeholder,
    grantedShares : Int,
    conditions : set Condition
} {
    no inputSecurity && one outputSecurity && no balanceSecurity
    outputSecurity in PlanSecurity
    no outputSecurity.parentSec
    outputSecurity.initialTx = this

    no s : Security | s.terminalTx = this
    all s : Security | s.initialTx = this implies s = outputSecurity

    outputSecurity.owner = issuedTo
    outputSecurity.grantedShares = grantedShares
    outputSecurity.conditions = conditions

    no parentTx

    zero[outputSecurity.vestedShares]
    zero[outputSecurity.exercisedShares]

    let x = sum c : outputSecurity.conditions | c.conditionedShares {
        eq[outputSecurity.vestedShares, x]
    }
}
```

**Listing 14: The plan issuance transaction signature**
The Plan Security Cancellation Transaction

This signature is directly analogous to the total stock security cancellation (Section 3.7.3). Figure 3.12 shows an example instance, and Listing 15 presents the Alloy source code.

Figure 3.12: An example instance of the plan cancellation transaction

```
sig PlanCancellationTx extends PlanTx {
}
```

Listing 15: The plan cancellation transaction signature

The Plan Security Exercise Transaction

The Exercise Transaction for plan securities adheres to the foundational principles and methodologies common to other transactions. However, it necessitates added accounting verifications to manage distinct metrics, such as exercised shares, vested shares, and granted shares. This increased scrutiny is required due to the intricate structure of plan securities.

Figure 3.13 illustrates a sample instance of this transaction, while Listing 16 provides the corresponding source code.

Figure 3.13: An example instance of the plan exercise transaction
The Plan Security Vesting Transaction

The Vesting Transaction for plan securities is the most intricate within the system. When the transaction occurs, any conditions that are triggered by the specified date or event are marked as satisfied and subsequently removed from the resultant, updated plan security transaction. As these conditions are fulfilled, the corresponding shares transition into a vested state, thereby becoming eligible for exercise. Apart from this specialized treatment of conditional triggers, the transaction aligns with the typical structure of other transactions, featuring one input and one output security.

The specific triggers and a presentation of the vesting system is provided in Section 3.8. Listings 17 provides the basic signature for vesting transactions.
The Plan Security Vesting Date Transaction

The Plan Security Vesting Date Transaction serves as an indicator that a specified date has elapsed, potentially satisfying certain triggers embedded within the conditions of the plan securities. Figure 3.14 and Listing 18 respectively showcase an example instance and the source code corresponding to this transaction signature.

Figure 3.14: An example instance of the plan vesting date transaction
The Plan Security Vesting Event Transaction

Analogously to the vesting date transaction, The Plan Security Vesting Event Transaction serves as an indicator that a specified event has occurred. Figure 3.15 and Listing 19 respectively showcase an example instance and the source code corresponding to this transaction signature.

Figure 3.15: An example instance of the plan vesting event transaction
sig PlanVestingEventTx extends PlanVestingTx {
  event : Event
}
{
  one inputSecurity && one outputSecurity && no balanceSecurity
  inputSecurity in PlanSecurity && outputSecurity in PlanSecurity
  outputSecurity.parentSec = inputSecurity
  outputSecurity.initialTx = this
  inputSecurity.terminalTx = this

  all s : Security | s.initialTx = this iff s = outputSecurity
  all s : Security | s.terminalTx = this iff s = inputSecurity

  outputSecurity.owner = inputSecurity.owner
  satisfiedConditions = { c : inputSecurity.conditions |
    ∧ isTriggerSatisfiedByEvent0[c.trigger, event] }
  parentTx = inputSecurity.initialTx

  pos[vestedShares]
  eq[outputSecurity.vestedShares, add[inputSecurity.vestedShares, vestedShares]]
  eq[outputSecurity.grantedShares, inputSecurity.grantedShares]
  eq[outputSecurity.exercisedShares, inputSecurity.exercisedShares]
  eq[vestedShares, sum[satisfiedConditions.conditionedShares]]
}

Listing 19: The plan vesting event transaction signature

3.7.4 Constraints

Beyond the constraints provided as part of signatures, we enforce a few more constraints as stand-alone facts to ensure that the securities and transactions are consistent with each other.

Two very important facts, orderingParentSec and orderingParentTx, ensure that the parent relations are acyclic by establishing the partial ordering between securities and securities and transactions and transaction. This is shown in listings 20 and 21.

As we establish that children are always \( \geq \) than their parents, we ensure that there are no cycles in the graph induced by the parentSec and parentTx relations. This is an example where Alloy’s expressiveness shines, as we can express the fact in a very concise manner.
By stating all the signatures and constraints we have defined the possible instances of the model. But the model is only interesting if it possesses the properties that we expect to hold. We will now define the properties that we expect to hold in the model.

### 3.8 Vesting system

The original OCF vesting system can handle at most a combination of dates and events. We support both triggers for vesting, but add operators for the conjunction, disjunction and negation of triggers. The vesting trigger for vesting after a date thus can be negated and express conditions that might happen before a date.

In our model, each PlanSecurity has a Condition, which has a Trigger and a number of shares conditionedShares. The triggers form a simple expression language with two base cases (dates and events) and 3 propositional logic operators. The metamodel for our improved vesting system is shown in Figure 3.16.
3.8.1 Conditions

The structure of a condition is straightforward and serves the purpose of associating a quantity of shares, denoted as \texttt{conditionedShares}, with specific triggers. The signature for such a condition is displayed in Listing 22. Its primary function is to link a specific number of shares—\texttt{conditionedShares}—with a \texttt{trigger} that, when satisfied, allows for the release of those shares. It’s important to note that the \texttt{Condition} signature is the one referenced within plan security transactions. The source code representation for the condition is also provided in Listing 22.

```plaintext
sig Condition {
  conditionedShares : Int,
  trigger : one Trigger
}
```

Listing 22: Signature of a condition.
3.8.2 Triggers

The triggers are designed to be straightforward signatures as well. A `_children` relation is introduced to guarantee that the trigger expressions remain acyclic (by pairing it with a constraint). The signature for a trigger is presented in Listing 23. Signatures for additional types of triggers are depicted in Listings 24 and 25.

```
abstract sig Trigger {
  _children : set Trigger
}
```

Listing 23: Signature of a trigger.

**Event and Date-related**

Basic triggers necessitate either the occurrence of an event or the passing of a specific date. These triggers are categorized as "after" triggers, meaning they are activated following the event or date in question. These triggers can also be negated and combined either conjunctively or disjunctively, as discussed in Section 3.8.2. This added flexibility significantly expands the scope of possibilities compared to the original model, especially since the negation of an "after" trigger effectively translates to "until." Listings 24 and 25 present code for both base triggers.

Given that the triggers constitute an expression language subject to recursive evaluation (as detailed in Section 3.8.3), these two basic triggers serve as the base cases for the evaluation process.

```
sig AfterEvent extends Trigger {
  afterEvent : Event
}
```

Listing 24: Signature of an AfterEvent trigger.
Propositional logic

In addition to the basic triggers, logical operators "AND," "NOT," and "OR" are introduced. As previously noted, the negation operator, when applied to an "after date" or "event," signifies "until." Through the use of conjunction, it becomes possible to express vestings that are conditioned upon both achieving a milestone and meeting a deadline. The original model was limited to expressing only conjunctions of triggers, lacking the capability for disjunction and negation. The source code for these extended trigger types is provided in Listings 26, 28, and 27.
sig Negation extends Trigger {
  inner : one Trigger
}

Listing 28: Signature of a Negation trigger.

3.8.3 Unrolled evaluation function

Alloy is limited in its ability to natively support recursion. To address this, we adopt a strategy of unrolling the recursion through varying depths until we encounter the base case, as exemplified in Listing 29. For the scope of this study, a tenfold unrolling of the function is deemed adequate. This unrolled function is employed within the isTriggerSatisfiedByEventN predicates, which are designed to verify the fulfillment of specific conditions. The methodology echoes techniques used in model finders for code synthesis by bounding loop depths.

In the source code, the base case—termed isTriggerSatisfiedByEvent0—serves as the point where events and dates are actually verified.
pred isTriggerSatisfiedByEvent0([t : Trigger, e : Event] { 
  (isTriggerSatisfiedByEvent_BASE_CASE[t, e] 
  ) or ( 
    t in Conjunction \( k k \) isTriggerSatisfiedByEvent1[t.conjL, e] \( k k \) 
    \( \leftarrow \) isTriggerSatisfiedByEvent1[t.conjR, e] 
  ) or ( 
    t in Disjunction \( k k \) isTriggerSatisfiedByEvent1[t.disjL, e] || 
    \( \leftarrow \) isTriggerSatisfiedByEvent1[t.disjR, e] 
  ) or ( 
    t in Negation \( k k \) not isTriggerSatisfiedByEvent1[t.inner, e] 
  )
})

// until...
pred isTriggerSatisfiedByEvent9([t : Trigger, e : Event] { 
  (isTriggerSatisfiedByEvent_BASE_CASE[t, e] 
  ) or ( 
    t in Conjunction \( k k \) isTriggerSatisfiedByEvent10[t.conjL, e] \( k k \) 
    \( \leftarrow \) isTriggerSatisfiedByEvent10[t.conjR, e] 
  ) or ( 
    t in Disjunction \( k k \) isTriggerSatisfiedByEvent10[t.disjL, e] || 
    \( \leftarrow \) isTriggerSatisfiedByEvent10[t.disjR, e] 
  ) or ( 
    t in Negation \( k k \) not isTriggerSatisfiedByEvent10[t.inner, e] 
  )
})

// Base case
pred isTriggerSatisfiedByEvent_BASE_CASE([t : Trigger, e : Event] { 
  t not in AfterDate 
  t in AfterEvent \( k k \) t.afterEvent = e 
})

Listing 29: Unrolled evaluation function.

3.9 Checks

The model’s integrity is validated through a comprehensive suite of checks, segmented into accounting, count, and structure checks.
**Accounting Checks**, elaborated in Section 3.9.1, perform arithmetic verifications pertinent to the quantification of shares.

**Count Checks**, discussed in Section 3.9.2, validate the accurate cardinality of instances for each type, providing substantial analytical utility despite their conceptual straightforwardness. For example, the quantity of `initialTx` relations should align with the number of securities.

**Structure Checks**, expounded in Section 3.9.3, examine the topological soundness of the securities and transactions graph, assuring, for instance, the absence of cyclical structures.

Model validation is facilitated by the Alloy Analyzer, which utilizes bounded-model checking by translating the Alloy model into an isomorphic SAT problem. In this SAT framework, each solution corresponds to a unique model instance. The Alloy Analyzer confirms the properties under specification by identifying counterexamples, thus offering a robust methodology for model verification.

### 3.9.1 Accounting checks

Accounting identities serve to validate the model. These identities are domain-derived and are implemented as checks within the model. These identities operate based on the following categorizations of shares at various junctures in the model.

In the context of plan securities, shares are classified into exercised, vested, and granted categories. The categorization for shares related to plan securities is delineated in Listing 30.
fun _exercisedShares : set Int {
  sum i : PlanExerciseTx | i.exercisedShares
}

fun _vestedShares : set Int {
  sum i : PlanVestingTx | i.vestedShares
}

fun _grantedShares : set Int {
  sum i : PlanIssuanceTx | i.grantedShares
}

Listing 30: Plan security shares

The issuance of new shares into the system is encapsulated by the issuedShares and newShares functions, as illustrated in Listing 31.

fun _issuedShares : set Int {
  sum i : StockIssuanceTx | i.issuedShares
}

fun _newShares : set Int {
  add[_.issuedShares, _grantedShares]
}

Listing 31: New shares

Shares that have been cancelled are represented by the cancelledShares function, as delineated in Listing 32.
fun _totallyCancelledShares : set Int {
    sum i : StockTotalCancellationTx | i.inputSecurity.shares
}

fun _partiallyCancelledShares : set Int {
    sum i : StockPartialCancellationTx | i.cancelledShares
}

fun _cancelledShares : set Int {
    add[_totallyCancelledShares, _partiallyCancelledShares]
}

Listing 32: Cancelled shares

Lastly, the measure of all shares in circulation, also known as outstanding shares, is encapsulated by the outstandingShares function, as illustrated in Listing 33.

fun _currentStockSecurities : set StockSecurity {
    StockSecurity - _expiredStockSecurities
}

fun _expiredStockSecurities : set StockSecurity {
    terminalTx.StockTx
}

fun _currentShares : set Int {
    sum s : _currentStockSecurities | s.shares
}

Listing 33: Outstanding shares

We can now delineate the accounting properties subject to verification. These properties are derived from domain equations:

\[ 0 \leq \text{exercisedShares} \leq \text{vestedShares} \leq \text{grantedShares} \quad (3.1) \]
\[ 0 \leq \text{cancelledShares} \leq \text{issuedShares} + \text{exercisedShares} \quad (3.2) \]

\[ \text{currentShares} \leq \text{issuedShares} + \text{exercisedShares} - \text{cancelledShares} \quad (3.3) \]

The checks are implemented as in Listings 34, 35 and 36.

```java
check {
    lte[0, _exercisedShares]
    lte[_exercisedShares, _vestedShares]
    lte[_vestedShares, _grantedShares]
}
```

Listing 34: Check plan shares

```java
check {
    lte[0, _cancelledShares]
    lte[_cancelledShares, add[_issuedShares, _exercisedShares]]
}
```

Listing 35: Check cancelled shares

```java
check {
    lte[_currentShares, sub[add[_issuedShares, _exercisedShares], _cancelledShares]]
}
```

Listing 36: Check outstanding shares

### 3.9.2 Count checks

Count-based verification is instrumental in precluding a large swath of invalid instances, bearing resemblance to techniques such as reference counting. These checks are computationally efficient and serve as valuable tools for model validation.

The count-based checks are formulated to evaluate the following equations:
\[ \text{terminal}T_x = \text{inputSecurity} \]  

(3.4)

\[ \text{initial}T_x = \text{outputSecurity} + \text{balanceSecurity} \]  

(3.5)

\[ \text{inputSecurity} + \text{balanceSecurity} + \text{outputSecurity} = \text{terminal}T_x + \text{initial}T_x \]  

(3.6)

\[ \text{parent}T_x \leq \text{Tx} \]  

(3.7)

\[ \text{initial}T_x \leq \text{Security} \]  

(3.8)

The source code for the checks is shown in Listing 37.
Structural verifications serve a pivotal role, confirming that the graph of securities and transactions constitutes a Directed Acyclic Graph (DAG), that no securities exist without corresponding transactions, and that all securities are anchored in an issuance transaction.

The resultant graph’s simplicity is intrinsically linked to its traceability, underlining the efficacy of the model in ensuring transparent and accountable transactions.

No cycles in graph

Alloy provides a dedicated graph module comprising predicates and functions tailored for graph analysis. Among these is the \texttt{dag} predicate, which ascertains whether a graph de-
derived from a binary relation constitutes a Directed Acyclic Graph (DAG). The model can be invoked as delineated in Listing 38. Utilizing the `dag` predicate allows for a succinct articulation and verification of the graph’s acyclic nature, as demonstrated in Listing 39.

```plaintext
open util/graph[Security]
open util/graph[Tx]
```

Listing 38: Opening the `util/graph` module for securities and transactions

```plaintext
check { dag[parentTx] } for 5
check { dag[parentSec] } for 5
```

Listing 39: Check that the `parent` relationships induce an acyclical graph.

**Graph roots**

We validate that the graph engendered by the `parentSec` and `parentTx` relationships forms a Directed Acyclic Graph (DAG). According to domain-specific principles, every share can be traced back to an original issuance. Therefore, each security invariably contains an issuance in its historical lineage. This invariant holds across multiple securities and extends to the base case involving a single security within a trace.

By transitive implication, if each security has an issuance in its lineage, the initial security must be valid. Subsequent securities in the sequence must either be issuances themselves or descendants of the preceding security, paralleling the logic of natural induction.

```plaintext
check {
    { all s : StockSecurity | some i : StockIssuanceTx + PlanIssuanceTx | i in s.*parentSec.initialTx
    } checking
    { all p : PlanSecurity | some i : PlanIssuanceTx | i in p.*parentSec.initialTx
    }
} for 5
```

Listing 40: Checking that all securities contain an issuance in their lineage
No overlap in portfolios

Within the given domain, it is axiomatic that each security can have a singular owner. Formally, this necessitates that all portfolios—defined as the aggregated securities belonging to an individual stakeholder—must be disjoint.

Listing 41: No overlap in portfolios

```plaintext
check {
    all disj s1, s2 : Stakeholder
    | no s1.portfolio & s2.portfolio
} for 5
```

3.9.4 Discussion and contributions

Systems managing security transactions, including capitalization table systems, necessitate both traceability to origin and adherence to accounting principles. Our formalized model, validated through Alloy, successfully fulfills these prerequisites. By articulating the properties in a precise manner, we facilitate effective communication with both domain experts and software engineers. Furthermore, employing Alloy’s verification capabilities significantly enhances quality assurance levels, which were not only lacking in the Open Cap Table Coalition’s original model but also inherently unexpressable using JSON Schema.

3.10 Discussion

3.10.1 Interesting aspects of the design

A few aspects of the model are worth mentioning, because they solve complicated implementation problems in a simple manner.

Mostly local constraints

Most constraints are stated locally, in the sense that they are stated in terms of the fields of a single transaction. This makes it easier to write validation code, because we can check each transaction in isolation. Global constraints are much harder to reason about.
Current state is always kept reified

The current state is always kept reified, avoiding the need to replay the whole history to get the current state. This is a big win, not only performance-wise but also in terms of simplicity: it makes the model much more understandable.

Each security issuance gives rise to a separate graph of transactions

It is clear that each security issuance gives rise to a graph of transactions that is separate. This can be a boundary of transactions in the sense of concurrency, since it identifies orthogonal partitions of the state. This allows concurrency, which aids both in performance and in high availability.

It is time to consider what have we gained by using Alloy to model the system.

3.11 What have we gained?

The original model had ambiguities in its interpretation and no properties of any kind specified. The state of the system had to be reconstructed by replaying all transactions, for which no unambiguous interpretation is available.

We have gained the following:

1. We have a formal model of the system that is unambiguous and that can be used to reason about the system.

2. Our model is practical from the point of view of implementation.

3. We can ensure that the model behaves correctly by respecting constraints over the bipartite graph of transactions and securities and over accounting restrictions.

4. The visualizations provided by the Alloy Analyzer are very useful to understand the model and to communicate it to others.

5. The current state of the system is now reified and easy to access.

Both were possible only after:

1. Carefully analyzing the original model and identifying the key concepts and mapping them conceptually as entities and constraints.

2. Establishing what properties define the system as correct.
3. Using model-checking and simulations to refine the model.

All three steps above require using Alloy as a modeling language. The kinds of properties and constraints we require are not expressible in JSON Schema, and we certainly do not enjoy the benefits of model checking and simulation without Alloy.
Chapter 4

Relation to Contracts and code normativity

The inherent difficulty of encoding legal norms into software stems from the ambiguity and complexity of legal language, coupled with the potential for context-dependent interpretation. Traditional contracts, although comprehensive, often rely on a broader legal framework to fill in gaps and resolve ambiguities, features that are challenging to replicate in code.

Our work engages with the field of code normativity, which aims to translate legal rules into executable code. We address similar challenges of interpretation and execution in the context of capitalization tables. By modeling essential contract elements in Alloy, we contribute to both cap table management and the broader issue of legal-to-code translation. Our approach serves as a practical case study for how formal methods can bridge the legal and technical realms, streamlining transactions and compliance.

4.1 Our Work in the Context of Code Normativity

In our model, we converted essential elements of contracts pertinent to capitalization tables into Alloy. By doing so, we not only serve the immediate purpose of better cap table management but also contribute to the more generalized problem of translating legal norms into executable software. Our model, therefore, serves as a case study in how formal methods can be applied to encode complex legal frameworks in a machine-readable format, thereby
facilitating automatic execution and validation.

4.2 Role of Contracts in Investment Activities

Investment instruments specific to start-ups, such as venture rounds and stock option plans, are formalized through contracts written in natural language. The execution of these contracts, after being ratified by all involved parties, activates various provisions and stipulations, including the allocation and pricing of instruments like shares or options.

4.3 Incompleteness and Reliance on Legal Frameworks

Traditional contracts, although exhaustive in defining roles, obligations, and privileges, are intrinsically incomplete. They function under the overarching structure of the law, which serves as the foundational contract dictating the rules of interaction. Oftentimes, contracts may include direct or implied references to legal clauses or other contractual documents to fill in the gaps in their stipulations.

4.4 The Challenge for Software Engineers: From Prose to Code

For software engineers responsible for developing cap table management systems, the narrative-driven contracts present an obstacle. Translating these contracts into functional data models and algorithms is neither trivial nor direct, given the lack of formal language and machine-readability in the contracts. The same legal construct might be represented in myriad ways across different contracts, adding to the complexity of translation.

4.5 Simplifying Contracts Through Our Model

Our computational model distills contracts to their essential elements, focusing on what is pertinent to cap tables:

- Stock issuances and both partial and total transfers correlate with share purchase agreements.
- Vesting systems and plan transactions can be represented by stock option plan grant awards.
- Option exercises correspond to stock option exercise agreements.

We do not posit that traditional contracts are obsolete; however, the amalgamation of formal specifications alongside traditional contracts would undeniably facilitate a streamlined execution process.

4.6 Code Normativity and Community Efforts

Two emerging trends within the community underscore the concept of code normativity in the context of financial transactions.

4.6.1 Smart Contracts

The notion of smart contracts, conceptualized by Nick Szabo in 1994, encapsulates computer protocols that automate, verify, or even eliminate the need for conventional contracts. The advent of Bitcoin propelled the development and adoption of smart contracts, with Ethereum pioneering the field by offering a general-purpose blockchain featuring an embedded programming language. This enabled the creation of smart contracts in a high-level language called Solidity, which is compiled to Ethereum Virtual Machine (EVM) bytecode.

However, Solidity was initially susceptible to security vulnerabilities, notably reentrancy attacks, which were exploited in the notorious DAO hack. The incident was not only financially devastating—resulting in the loss of 3.6 million Ether—but also socially divisive, as it led to an Ethereum hard fork.

In contrast, our model not only captures individual contracts but also models the global dynamics of the system, which individual smart contracts often overlook.

4.6.2 Tax and Accounting: The Catala Approach

The second trend focuses on leveraging formal languages for tax and accounting, exemplified by Catala, a domain-specific language. Catala facilitates the definition of rules in a manner that is both interpretable by humans and machines. A notable feature of Catala is its ability to identify contradictions within the rules, addressing a common issue in the domain of tax and accounting.

Similarly, our use of Alloy serves analytical purposes, allowing us to examine the collective consistency of various contracts and rules. This analytical power aligns our work closely with the spirit of the Catala approach.
By harnessing formal languages like Alloy, we aim to bridge the gap between natural language contracts and computational representations, thereby contributing to the broader discourse on code normativity in financial systems.

4.7 An example rendition of our model as prose contracts

In this section, we illuminate the process of outlining a stock option plan utilizing an instance of an Alloy model, showcased in Listing 42.
pred sop1 {

  one p : PlanIssuanceTx {
      p.strikePrice = 10

      // Vesting Schedule
      one c : p.conditions | c.trigger in AfterDate && c.trigger.afterDate = Date1 and
          eq[c.shars, 20]
      one c : p.conditions | c.trigger in AfterDate && c.trigger.afterDate = Date2 and
          eq[c.shars, 20]
      one c : p.conditions | c.trigger in AfterDate && c.trigger.afterDate = Date3 and
          eq[c.shars, 20]
      one c : p.conditions | c.trigger in AfterDate && c.trigger.afterDate = Date4 and
          eq[c.shars, 20]

      // A bonus of 10 shares is issued if the share price is greater than £20 by Date2
          (Event1)
      one c : p.conditions | c.trigger in AfterEvent && c.trigger.afterEvent = Event1 and
          eq[c.shars, 10]

      // A bonus of 10 shares is issued if the share price is greater than £30 by Date2
          (Event1 again)
      one c : p.conditions | c.trigger in AfterEvent && c.trigger.afterEvent = Event1 and
          eq[c.shars, 10]

      #p.conditions = 5
  }
}

Listing 42: Predicate describing a stock option grant

This model is further converted into a closely analogous prose rendition, offering an illustration of how the logical and structured format of an Alloy model can be translated into the legal language used in contractual agreements.
STOCK OPTION AGREEMENT

This Stock Option Agreement (the "Agreement") is made and entered into as of [Date], by and between [Company Name], a [State of Incorporation] corporation (the "Company"), and [Name of Optionee], an employee of the Company (the "Optionee").

Grant of Option. The Company hereby grants to the Optionee an option (the "Option") to purchase a total of up to 60 shares of common stock, $10 strike price per share, of the Company (the "Shares") on the terms and conditions set forth in this Agreement.

Vesting Schedule. Subject to the terms and conditions set forth herein, the Option will vest in accordance with the following schedule:

a. 20 Shares shall vest on [Date1];

b. An additional 20 Shares shall vest on [Date2];

c. An additional 20 Shares shall vest on [Date3];

d. An additional 20 Shares shall vest on [Date4].

Additional Vesting Conditions.

a. A bonus of 10 additional Shares shall be issued to the Optionee if the closing price of the Company’s common stock is greater than $20 on [Date2];

b. An additional bonus of 10 Shares shall be issued to the Optionee if the closing price of the Company’s common stock is greater than $30 on [Date2].

While this example is not exhaustively detailed, it demonstrably highlights the feasible potential that tools might aptly employ our model to specify particular instances and proficiently generate legal text, contingent upon their validation. This facilitates the seamless transition from structured Alloy models to comprehensive and legally sound prose contracts, enhancing the efficiency and efficacy of contract creation and management.
Chapter 5

Conclusion

This thesis commenced with an examination of a data-centric domain model, endowed with extensive business logic and domain-specific knowledge, yet constrained by its technological underpinning. It was posited that the Open Cap Table Format (OCF) is inherently limited in its computational capabilities; its constraints are particularly manifest in its inability to be fully specified using JSON Schema. While the OCF excels in data format validation—consonant with its role as a data interchange format—it lacks the mechanism for managing capitalization tables in a semantically rigorous manner.

By employing Alloy, a lightweight formal modeling language, this research has led to the development of a more robust model capable of calculating the state of a capitalization table, validating transaction and security input data, and subjecting the model to automated validation for ensuring its correctness. Contrary to the original model, the resultant model facilitates exhaustive tests concerning business logic.

Transitioning from a data specification to a domain specification, this research took the implicit domain knowledge of the original model as its foundation. It then identified and formalized the key abstractions in Alloy, effectively moving from a syntactic to a semantic model that accounts for relationships between different types of entities and expected invariants based on domain knowledge.

The benefits accrued from this transition include:

1. Computational ability to derive the state of a capitalization table given a set of securities and transactions.
2. Preservation of immutable data, akin to the original model, while ensuring a readily computed state.

3. Assured traceability back to the original issuance of shares for any given security.

4. Implementation of accounting constraints to prevent the erroneous creation or elimination of shares.

5. Introduction of count checks for additional validity by eliminating implausible graphs.


7. Enhancement of the vesting system to express a broader range of conditions through logical operators.

Consequently, this research not only offers a system specification for capitalization tables but also lays the foundation for formalizing financial transaction contracts. The resulting transaction constraints, when articulated as contract clauses, yield a representation that is more explicit, rigorous, and concise than conventional legal language, focusing particularly on operational facets of contracts.

Moreover, as shown in Chapter 4, our work has implications for the field of code normativity, which are reflected in the future work as opportunities for contract generation from Alloy or other formal models.

5.1 Future Work

The research presented herein serves as an initial platform for continued exploration into the application of lightweight formal methods in financial software design. Possible extensions and directions for future work are:

- **Model Extension:** Incorporate additional securities and transaction types.

- **Temporal Modeling:** Utilize the temporal logic capabilities introduced in Alloy version 6 for constructing state-machine-based capitalization table models.

- **Automated Contract Generation:** Investigate the feasibility of auto-generating textual contracts through Alloy-to-text translators, thereby eliminating the laborious manual verification process.

- **Code Generation:** Explore automated code generation using Alloy’s Java API.
Given most business logic is succinctly encapsulated as constraints, the target programming language would require minimal features to support the generated code.

- **AlternativeFormalisms:** Examine the applicability of other formal methods, such as TLA plus, for specifying similar problems from different perspectives.
Bibliography


