A natural solution to the problem of real-time monitoring and management of structured counterparty exposure limits

Seán Matthews
Financial Markets,
IBM Unternehmensberatung GmbH,
Lyoner Str. 13, 60528 Frankfurt am Main,
Germany.
+49 (0) 1 70/7 85 97 28
`Sean.Matthews@de.ibm.com`

May 31, 2002

Abstract

We describe a logic (based on Boolean algebra)—together with supporting algorithms and domain specific optimisations—for monitoring and managing counterparty risk, and in particular, for implementing high speed pre-deal limit checking (our logic should also be applicable to any similar application where transactions must be classified in real-time). Our design is general, fast, and insensitive to scaling. We provide some analysis of complexity, which we relate to empirical aspects of the problem. We also analyse the problem of how to implement well behaved reservations. Finally we describe an application to automatic limit mangement.

Contents

1 Introduction

In this paper we describe a fast and flexible solution we have developed to a core problem in the design of systems for monitoring (and managing) counterparty risk, and other similar applications which have to classify transactions in real time. Our solution, so far as we can ascertain from examination of user manuals and sales materials, is significantly more flexible, and faster, than those offered in systems currently available for purchase in the market, and removes a core scalability constraint. Further, we argue that our solution is 'natural', on the grounds that it is based on a canonical logical language, and that this language represents a probable upper bound in terms of both expressiveness and speed.

The problem, called *transaction allocation* (or *pre-deal limit checking*), is as follows: The risk control officer of a trading operation wants to be sure that the total exposure of his operation is safely diversified. To this end, he defines limits against a structured set of *consolidation points* in his *exposure monitoring system* (EMS), against which all transactions are classified. He may also want to monitor, though not limit, more specific consolidation points, to help him actively manage the limits he has specified. Consolidation points might include simple categories such as 'car manufacturers', 'financial companies', 'energy utilities', etc., but may also include more complex classifications such as 'non-German European car manufacturers', and possibly very complex categories such as, e.g., transfer risk of various sorts.[1] Each trade is submitted for approval to the EMS, which, before authorising it, checks against all defined consolidation points (performs a pre-deal limit check), to be sure that no defined limits would be exceeded as a result. Since an immediate response to the check is necessary (a trader will not wait half an hour) the EMS must be able to identify immediately all the consolidation points relevant for the deal from among the tens of thousands that it maintains.

Transaction assignment is clearly not trivial: we have to reconcile, on the one hand, a language flexible enough to formalise all the consolidation points the risk control officer needs, with, on the other, one which allows us to find instantaneously all the consolidation points relevant for a deal.

1.1 Our contribution

We claim that there is in fact a 'natural' solution to the problem, based on Boolean algebra. We argue that:

- Boolean algebra provides a flexible, expressive language, more than capable of capturing the consolidation points a risk manager needs.

[1]Transfer risk could be incurred in, e.g., a deal with a counterparty with head office in country X, where neither of the two offices (party and counterparty) between whom the deal is closed, are located in X.

- That Boolean algebra represents a 'natural' upper bound to the class of practical lanuages for this application (with some interesting provisos for modal extensions, which we briefly discuss).
- That the resulting system is not only extremely fast, but insensitive to scaling. More precisely, the system should be able to classify deals against consolidation points in
  - (approx.) $O(\sum_i \log |J_i|)$ time, where J_i represents a partition of the set of basic properties used to define consolidation points. This follows not directly from the algorithm, but from assumptions about the properties of sets of consolidation points.
  - in practice, a few hundred thousandths of a second, assuming a respectable implementation on modern hardware.

The risk with the solution we propose is possibly exorbitant space requirements, therefore an important part of our contribution is the series of optimisations which we use to keep the space requirement under control.

We would be surprised if the algorithm we propose is not already documented in the literature, and we make no great claims to originality for it. We see our contribution rather in our exploration of the issues that arise in the attempt to apply it to the problem at hand; i.e. automation of limit management, and in particular, how to deal with time, how the various components should fit together, our domain specific optimisations, and maintenence issues, etc.

1.2 Structure of paper

The rest of this paper is structured as follows: in §2, we provide background (e.g. we describe the environment in which an EMS is located) and other general information; in §3, we discuss alternatives for a language of consolidation points, and argue that Boolean logic is the natural candidate. In §4 we present the core algorithm, however in order to make this useable we need to introduce some optimisations based on various properties of sets of structured limit points; we present these optimisations, together with remarks on performance and scalability in §5. In §6 we then consider how to treat the temporal concepts (e.g. *long term*, *short term*, etc.), and motivated by the example of modal (temporal) logic, consider general extensions to our model, which we dismiss, on various grounds. In §7 we discuss how we can modify the set of consolidation points of a running system. In §8 we consider the interesting question of how reservations behave, and discuss the issues that have to be addressed in an implementation. In §9 we present a simple example of how limits can be actively managed. Since we recognise that no formal model can be complete, in §10 we consider ways to deal with concepts which do not naturally fit into our model. Finally, in §11, we attempt to make comparisons with commercial solutions with which we are familiar, summarise, and draw conclusions. It was not possible to provide a full survey of relevant related work, since related work is commercial software, to which we have not had access; we have however provided some remarks based on publicly or quasi-publicly available information, such as sales presentations.

2 Background

Before proceeding to deal allocation, it is useful to describe in more detail the environment in which an EMS is located. The system communicates with four distinct actors:

- the *traders*,
- the *risk manager*,
- *static data*, and
- *external computing resources.*

The traders can ask the system if a particular deal is currently permitted, and after entering into a deal, confirm that they have done so. (Note that these are two separate acts: the query can reserve resources in the system guaranteeing that the deal is possible, but this reservation will be cancelled unless a confirmation of the deal is received within a certain period.)

The risk manager not only manages the exposure limits for the system but is also able, for instance, to reserve facilities for particular traders (e.g. he might assign \$100M of a possible \$150M of possible exposure to South-american banks, so that other users are able to take a maximum position of \$50M, irrespective of how much of the reserved \$100M has actually been used). He also receives regular reports about current exposure, and may submit *ad hoc* queries to extract information that is not part of the regular consolidation point set; e.g. exposure to firms owned by Japanese banks. He may also introduce at any time *ad hoc* limits on particular types of exposure (the most obvious example of this is that he may want to suspend immediately all business with some class of counterparty.

The static data system is the third important actor. Not all relevant information for a deal is provided directly as part of the transaction that the EMS receives; rather, this information must be obtained indirectly, by querying the static data system. Thus, for instance, a limit may have been placed on exposure to a corporate group A. The EMS, however, cannot tell, just from the information it directly receives, whether a deal implies exposure to A; all that that information says is that the deal is with company B. Reference to static data is necessary find out whether B is in fact a part of company group B.

Finally, the EMS may also be connected to external computing resources which it may use to perform demanding calculations in parallel. These external resources are not directly relevant for the discussion in this paper, but any considerations of scalability have to take them into account. (If complex exposure calculations are performed locally, then the scalability of the system is clearly fundamentally different than if those calculations are farmed out to be done in—more scalable—parallel.)

3 Candidate languages for formalising consolidation points

3.1 *Ad hoc* languages

There are two different ways we can go about finding a suitable language for consolidation points: either we can invent something *ad hoc* ourselves (in practice, the design of *ad hoc* languages is usually driven by the design of the GUI, which in turn is provided by the business-side analysts as part of the inital specification for the system), or we can adopt a canonical—that is well understood and tested—language from the general literature of mathematics. This paper argues, as we have already made clear, that a canonical language (Boolean algebra) is the most suitable choice, but to support this claim we must also argue that the development of *ad hoc* languages is unlikely to produce a more effective solution.

The category of *ad hoc* solutions itself can be further divided into two possibilities where we define consolidation points by

1. some arbitrary set of recursive predicates, or
2. by form filling of the sort that we find in much commercial software, where the architecture is driven by the graphical interface.

We can immediately dismiss 2, since not only does it leave the suspicion that it results in the dangerous situation where consolidation points are defined according to what the system allows, rather than what is necessary, but also because it is properly less expressive than the solution we describe below. The argument against 1 is more subtle, since by definition it covers any possible solution, including the one we propose; to argue against it we have to argue that it is impossible, or at least improbable, that a better solution than the one we propose exists.

We could make the historical argument that the likelihood that a better language exists is negligeable: if it existed, then it would already have been investigated by mathematicians. But we can considerably strengthen our case by pointing out that there is simply little or no room in the space of possibilities for such a language. Below we propose two candidate languages from the canon of logic (Boolean algebra and predicate logic), and after examining them both, we dismiss predicate logic for various reasons in favour of the apparently weaker Boolean algebra, which in turn we show to be extremely effective for our application (see the claims in the introduction, or the formal arguments below). A competitive *ad hoc* solution would have to be both comparably efficient (it need not to be quite so efficient as Boolean algebra, which, we argue, has performance to spare) and also both more expressive—since we establish Boolean algebra as a reasonable lower bound), and differently expressive than predicate logic (it cannot include predicate logic, since then it would be vulnerable to the same arguments). The obvious class of languages satisfying this requirement is that of prepositional logics extended with intensional (modal) operators. But this is itself a more or less well defined canonical class, and therefore, by definition, not *ad hoc*, and it is not clear that a modal logic brings much useful extra expressiveness for the current application.[2] At this point we are left without much further room in which to search for a candidate, and now we again suggest if such an expressive language existed there, then it would already have been formalised and investigated.

3.2 Canonical languages

The literature provides us with two canonical languages as candidates for a specification language: Predicate logic and Boolean algebra, the second being (*via* isomorphism with propositional logic) a proper subset of the first.

3.2.1 Predicate logic

Consider predicate logic first: the language of $\forall$ (*forall*), $\exists$ (*exists*), $\neg$ (*not*), $\wedge$ (*and*), $\vee$ (*or*), $\rightarrow$ (*implies*). This is a possibly unfamiliar form of a familiar notation, since it corresponds, via relational algebra, to the language of relational databases.[3]

A typical possible consolidation point for our system might be

> *Deals with a company which is part of a group with interests in the oil industry*

which we can formalise in predicate logic simply as:

$$\{\ d \mid \exists c.\text{company}(d, c) \wedge \exists g.\text{partof}(c, g) \wedge \text{industries}(g, \text{oil})\ \}$$

which translated back into ordinary language defines the set of deals d such that the company associated with d is c, and c is part of a corporate group g with significant interests in the oil industry.

[2]Modal logics are also usually computationally less tractible. For further discussion, see §6 about time judgements, as well as remarks in the appendix.

[3]Note that when we say 'relational database' we are talking in the pure formal sense without, e.g., transitive relations, which can be calculated with SQL queries but are not part of the relational model *per se*.

In relational algebra, the same class might be defined:

$$\mathrm{Join}_{\mathrm{companies},1}(\mathrm{transactions}, \mathrm{Join}_{2,1}(\mathrm{partof}, \mathrm{industries}))$$

Relational algebra is familiar, and expressive; why should we reject it here? The problem is that we have the inverse to the usual relational database problem: instead of

find all the transactions which match a query,

we need

find all the queries which match a transaction.

Unfortunately, in the general case, this is both a hard problem, and one which relational databanks are not designed to solve. Rather than consider the general question, we can ask, what performance could we expect from a system which contains a single consolidation point, defined as above; i.e., if a deal D enters the system, how long does it take to tell whether the deal belongs to the consolidation point some D, whether

$$\exists c.\mathrm{company}(\mathrm{D}, c) \wedge \exists g.\mathrm{partof}(c, g) \wedge \mathrm{industries}(g, \mathrm{oil})$$

or, again in relational algebra terms,

$$|\mathrm{Join}_{2,1}(\mathrm{Select}_1(\mathrm{company}(\mathrm{D}), \mathrm{partof}), \mathrm{industries})| > 0.$$

However Select takes time $O(N)$ and Join time $O(N \log N)$ in the size of their arguments,[4] thus to calculate whether a transaction is captured by a single relational query is linear in the sum of the sizes of the tables to which it refers (and in practice usually worse), and we would have to run the same check for each consolidation point contained in the system.

What we need is a way to invert the traditional relational algebra problem. We argue in the next sections that in the current (though not in the general) case it is indeed possible to invert the problem, and that by making a few small, reasoned, compromises, it is possible to use this theoretical argument as the basis of a practical system.

3.2.2 Boolean algebra

Boolean algebra, is, via a well-known isomorphism with propositional logic, the fragment of predicate logic without the quantifiers $\forall$ and $\exists$, or variables. In Boolean terms, we simply read the propositions (more accurately, ground instances of atomic predicates) as sets, and the logical connectives as set theoretic operations: $A \wedge B$ becomes intersection, $A \cap B$; $A \vee B$ becomes union, $A \cup B$; and $\neg A$ becomes set complement, $-A$. The result is a very simple language; before considering whether it is sufficient for our purposes, we should ask if it has even the necessary properties. In fact we have:

[4]Note that we can sometimes improve the performance of the operations if we can anticipate the particular circumstance where the relation will be used.

1. Given sets A and B selected from a universe of transactions Υ, we can construct the set corresponding to the boolean formulae $A \cup B$ and $A \cap B$ in time $\mathrm{O}(|A|+|B|)$, and to $-A$ in time $\mathrm{O}(|\Upsilon|)$; i.e. we can construct the set corresponding to any Boolean formula in time proportional to the number of connectives times the number of transactions.

2. Given a set of sets Λ constructed using the standard boolean functions over sets Θ_i^j, where $j \neq j' \rightarrow \Theta_i^j \cap \Theta_i^{j'} = \emptyset$, in a universe of transactions Υ, then given any x in Υ, we can construct the set $\{ \gamma : \Lambda \mid x \in \gamma \}$ in time (approx.) $\mathrm{O}(\sum_i \log N_i)$ where N_i is the number of distinct Θ_i^j for i.[5]

(1) is obvious; we show and apply (2) below (§4.2). The result implies that a solution based on Boolean algebra is both extremely fast, and also essentially insensitive to scaling.[6]

3.3 The expressiveness of Boolean algebra

We have argued that predicate logic (i.e. relational algebra) is a powerful language, allowing natural language descriptions of consolidation points to be mapped essentially directly into formal specifications. In general, predicate logic is more expressive than propositional logic (i.e. Boolean algebra), but in fact certain properties of the information we are handling here means that in fact this increased expressiveness is illusory: any predicate expression can be reduced to a propositional one.

There are two sides to this point, of course: the technical argument, and its practical implications. Though the technical result is not directly useful, it provides the intuitions for more practical machinery.

3.3.1 Reducing predicate logic in an EMS to boolean algebra

The reduction follows from the fact that the only unbounded domain of objects with which we are concerned is the set of incoming transactions, all other domains (companies, company groups, industries, etc.) are completely defined and fixed (i.e. part of the *Static* data). Further, while we may quantify over the domains in the static data (as we do in the examples above), we never want to quantify over the set of transactions.

We also assume the static data is defined in terms of binary relations. This reflects our experience of real static data, and makes the analysis much

[5]This result is not, in general, true: it depends, as will be seen, on the fact that we are able to decide between the Θ_i^j themselves in log time; i.e. essentially, to exploit fast associative arrays.

[6]I.e., To double the response time, we would have to *square* the number of consolidation points occuring in each judgement class

simpler, though it is not absolutely necessary.[7]

observe that we can assume that the static data consists only of binary relations: if the static data includes, e.g., a triple $R(a,b,c)$, we can reduce this to binary relations R_{12}, R_{13}, R_{23}, where $R(a,b,c) \leftrightarrow R_{12}(a,b) \wedge R_{23}(b,c) \wedge R13(a,c)$.

We start with an arbitrary query $P(d)$, where d varies over transactions. The first step is as follows: we first move all negations inwards to atomic relations, and then eliminate them by transforming the relations to their complements, then we observe that we can always eliminate quantification over a fixed, finite domain $\Delta = \{\delta_1, \ldots, \delta_n\}$, by the following transformations:

$$\forall x : \Delta . P(x) \equiv \bigwedge_{\delta_i \in \Delta} P(\delta_i)$$
$$\exists x : \Delta . P(x) \equiv \bigvee_{\delta_i \in \Delta} P(\delta_i)$$

Thus we can reduce any query to a quantifier and negation free proposition in one variable d, where that one variable stands for the transaction.

The next step is to reduce this proposition to disjunctive normal form; i.e.

$$\bigvee_i \bigwedge_{1 \leq j \leq n_i} P_{ij}$$

We assume that d appears free in each disjunct, since if a disjunct does not contain d it is either constant *true* in which case all transactions belong to the consolidation point, or constant *false* in which case the disjunct can be ignored.

Consider a single conjunction in d. This can be factored into a collection of (not necessarily disjoint) subconjunctions of the form:[8]

$$P_{i,c_1 \ldots c_n}(d) \equiv [\neg] P_{i1}^{[-1]}(d, c_1) \wedge \bigwedge_i [\neg] P_{i2}^{[-1]}(c_i, c_{i+1})$$

together with a single conjunction (disjoint from the above) of the form

$$\bigwedge_i [\neg] P'_i(d,d)$$

(since we are assuming only binary relations in the static data).

Finally, we observe that, since there are only a finite, known, number of relations and constants in the static data, there can be only a finite, known,

[7]We can reduce an n-ary relation R_n to n binary relations R_2^i by simply defining $R_n(a_1, \ldots, a_n) \leftrightarrow \bigwedge_{1 \leq i \leq n} R_2^i(a_i, a_1 \ldots a_n)$. R_n can then be reconstructed by application of $n-1$ joins.

[8]Where $[\neg] P^{[-1]}(a,b)$ denotes the negated or non-negated form of either $P(a,b)$ or $P^{-1}(a,b)$

number of distinct predicates $P_{i,c_1 \dots c_n}$, the same applies to $P'_1(d, d)$. If we assume these are made available as the names of constant sets, then we can translate the conjunction into the form:

$$\left(\bigwedge_i d \in P_{i,c_1 \dots c_{n_i}} \right) \wedge \left(\bigwedge_j d \in P'_j \right)$$

which is the same as

$$d \in \left(\bigcap_i P_{i,c_1 \dots c_{n_i}} \right) \cap \left(\bigcap_j P'_j \right).$$

We are here treating only one conjunct of the normal form proposition for the transaction, we thus finish by taking the union of the sets corresponding to each disjunct.

To summarise, for any static data set defined as sets $SD_{\text{Predicates}}$ of predicates, we can find an equivalent collection of sets SD_{Sets} of transactions, where the set of consolidation points defined in terms of $SD_{\text{Predicates}}$ using predicate logic is the same as the set of consolidation points defined in terms of SD_{Sets} using boolean algebra.

3.3.2 A practical version of the reduction

The practical drawbacks of the reduction above are obvious: the terms produced by the translation are going to be enormous, as is the number of sets we need. We thus now modify the idea to be useful in practice; the modified version does not try to be complete, but only to provide the facilities that are useful in practice.

First, we observe that we are not usually interested in the details of what particular points we go through in a path $c_1 \dots c_n$, we just want to know that the path follows the right *sort of* route, and where it ends up; e.g. we want to know that a transaction is with a company which is part of a group with interests in the oil industry, we are not, as such, interested in the company or the group, but only that there is some path

$$\begin{tikzcd} d \arrow[r, "\mathit{company}"] & \cdot \arrow[r, "\mathit{partof}"] & \cdot \arrow[r, "\mathit{industries}"] & \mathit{oil}. \end{tikzcd}$$

Further experience tells us that we have no interest in predicates for paths that go through negated relations: in practice we are only interested in the 'negation' of the whole path (i.e. if we want to know if it is not the case that the transaction is with a company which is part of a group with interests in the oil industry).

In short, the sets that we need are defined by simple paths to various points in the static data. We call these sets *simple* consolidation points because they are the components out of which all other consolidation points are constructed, and say that they correspond to basic *judgements* on a transaction. In fact there turn out not to be that many classes of simple consolidation points: in a prototype implementation, we identified approximately 30 of hem, including:

1. *Counterparty location=place*
2. *Counterparty headoffice location=place*
3. *Counterparty group headoffice location=place*
4. *Counterparty industries=industry*
5. *Counterparty group industries=industry*
6. *internal book=book ID*
7. . . . etc.

Note that these are not predicates, parameterised over, e.g., *place.* They are constant sets with these names. Thus, for instance, there is a set *Counterparty location=Paris*; there is no *Counterparty location* predicate as such.

From an implementation point of view, the nicest property of the judgement associated with a simple consolidation point is that making it almost always amounts simply to following arrows through the static data—something we can do extremely quickly. Thus, e.g., we can look up to compare the legally registered head-office of the company group of the external counterparty to a transaction by

$$\begin{tikzcd} d \arrow[r, "\textit{counterparty branch}"] & \cdot \arrow[r, "\textit{company}"] & \cdot \arrow[r, "\textit{part of}"] & \cdot \\ & \arrow[r, "\textit{group head}"] & \cdot \arrow[r, "\textit{main office}"] & \cdot \arrow[r, "\textit{location}"] & X \end{tikzcd}$$

i.e., from the transaction, get the direct counterparty, and from that look up the company, then the larger company group, then the controlling company, which has a head office, which has a location X, which is what we are looking for. In this case, all the pointer following is even deterministic.[9]

3.4 The language of consolidation points

We thus summarize the analysis in this section by describing the language that we provide for consolidation points.

We identify the paths through the static data that we might be interested in, and we define a class of sets for each of these paths, which we call simple consolidation points. With each consolidation point is associated a

[9]Clearly not all arrows are deterministic redirections, but in practice almost all are; deterministic arrows correspond to to the exclusive judgements discussed in §4.2.1.

judgement which, when applied to a transaction, returns *true*, or *false* ('is a member of this consolidation point' or 'is not a member').

We can then construct other *complex* consolidation points CP out of the provided simple consolidation points SCP as follows:

$$CP ::= SCP \mid -CP \mid CP \cap CP \mid CP \cup CP$$

(note that we do not restrict complex consolidation points to being in disjunctive normal form).

3.5 The expressiveness of our language

There remains the question of how well the language we have defined compares to a 'real life' risk mangement regime—even irrespective of how effectively it can be implemented. In fact as a test of our language we have worked systematically through the set of consolidation points used by a major German bank to manage counterparty risk, and have been able to formalise everything without problems. This does not mean that we are able to formalise anything at all, and we do consider some exceptions which may cause problems in §10.

3.6 Automatically maintained consolidation points

An important assumption for the practicality of the model we have developed here is that certain consolidation points may be maintained automatically. There are two reasons why this may be necessary, either to reflect the organisation of the static data, or to maintain syntactically complex standard definitions.

Consider these one at a time; first the maintanence of consolidation points which reflect hierarchies in the static data.

3.6.1 Hierarchies

Consider the classification of instruments in the system, which may represent a tree (or even just a well ordering); e.g. the category of *bonds* is made up of *fixed rate bonds*, *floating rate bonds*, etc. which in turn break down into, e.g. *zero coupon bonds*, etc., with the leaves corresponding to the specific sorts of instruments that actually identified as parts of transactions. This hierarchy can be encoded in terms of our logic here, simply as a set of definitions; thus we have:

$$\begin{aligned} \textit{bonds} &\equiv \textit{fixed rate bonds} \cup \textit{floating rate bonds} \cup \ldots \\ \textit{fixed rate bonds} &\equiv \textit{zero coupon} \cup \ldots \\ &\vdots \end{aligned}$$

Presumably this hierarchy is maintained automatically by some central resource in the bank that is also used for other front, back and middle office systems. It therefore makes no sense for the same definitions to be manually maintained in the limit sytem. Another example might be countries: since transactions are associated with offices which are in cities, there is—at least in theory—no need for countries as primitive concepts; instead we could define them as counjuncts of cities; e.g.

$$\begin{aligned}&\mathit{Direct\ counterparty\ France} \equiv \\ &\qquad \mathit{Direct\ counterparty\ Paris} \cup \mathit{Direct\ counterparty\ Lyon} \cup \ldots\end{aligned}$$

3.6.2 Other maintained defintions

More importantly, however, there are definitions which are simply too intricate to be maintained by hand. One example of this that we have encountered is the consolidation point collecting all transactions together where the head office of the external party to the transaction is located in a country with risk rating X and the internal party is located in a country with risk rating other than X. Since we are working in language which does not allow quantification over a domain, we have to expand this out to every possible value (essentially in the way suggested in §3.3, above). This is not formally complex, but it is syntactically intricate, and it is unreasonable to expect a user to maintain this manually as ratings categories change.[10]

4 The initial algorithm

In this section, we present the algorithm (or, more accurately, we describe how to construct the classification tree: in fact, given this, the run time algorithm is trivial). We should immediately add, however, that the algorithm in the pure case is probably useless, due to its space complexity; and also because of especially bad behaviour related to the nature of the particular problem at hand. To deal with these problems, we first suggest a modification of the basic algorithm, then, in §5, propose some important optimisations.

We assume a language of consolidation points as defined above, in §3.4. We can gather all the defined consolidation points and their names together into a single relation:

$$D : CP \times E.$$

The problem of deal assignment is then to find the names of all the consolidation points to which a transaction belongs.

[10] An interesting question is whether it would be worth the effort of providing such quantification, to be unfolded for definitions, as part of the language of consoliation points provided to the user; currently, we envision such definitions being maintained by code hardwired into the appropriate part of the static data.

4.1 The binary version

Step 1

We start by observing that every consolidation point has an equivalent normal form CP':

$$CP' ::= \bigcup_i \bigcap_{1 \le j \le n_i} [-]A^i_j$$

(i.e. a disjunction of conjunctions of possibly negated basic consolidation points) and that, as a result, D can be transformed into an equivalent form not just in $CP' \times E$, but in

$$D^* : \bigcap_j [-]A_j \times E$$

since we can replace a pair of the form

$$\Big(\bigcap_{1 \le i \le m} \bigcup_{1 \le j \le n_i} [-]A^i_j, \text{name}\Big)$$

in D^* with

$$\begin{gathered}\Big(\bigcap_{1 \le j \le n_1} A^1_j, \text{name}\Big)\\ \vdots \\ \Big(\bigcap_{1 \le j \le n_m} A^m_j, \text{name}\Big)\end{gathered}$$

without changing the meaning of D^* (we have factored a single consolidation point name into parts, but clearly a transaction belongs to the consolidation point name precisely if it belongs to one—or more—of the disjunctive components of that consolidation point). We say that this last form of D^* is in *normal form.*

Step 2

We can now generate the classification tree $CT(D^*)$ to assign deals to consolidation points from a normalised consolidation point/name relation by a recursive analysis.

1. Assuming that there is some non-empty conjunction in the list of pairs in D^*, choose some basic consolidation point A that occurs in one of these, and we build two new relations D^*_A and D^*_{-A} as follows:

$$\begin{aligned} D^*_A &= \{\ (R/A, n) \mid (R, n) \in D, (-A) \notin R\ \} \\ D^*_{-A} &= \{\ (R/(-A), n) \mid (R, n) \in D, A \notin R\ \} \end{aligned}$$

where R/A is just the conjunction R with A, if it occurs, deleted (if A is the only conjunct, we can denote the resulting empty conjunction by $\top$). Then we have:

$$CT(D^*) \quad = \quad \begin{tikzcd} & A \arrow[dl, "\text{false}"'] \arrow[dr, "\text{true}"] & \\ CT(D^*_{-A}) & & CT(D^*_A) \end{tikzcd}$$

2. If all the relations in D are of the form $(\top, \text{name})$, then $CT(D^*)$ is just
$$CT(D) = \{\ n \mid (\top, n) \in D\ \}$$

4.2 Adding multi-way branches

We now start to modify this basic idea for an algorithm to our particular circumstances. The basic idea simply assumes that we have a collection of unrelated sets of transactions $(A, \ldots, Z)$, which we combine together. However, this is not really the case. In reality, what we have is essentially a disguised quantifier free predicate logic, where groups of sets correspond to different instantiations of the predicate. Thus we have a group of sets corresponding to different counterparties, different countries, different industries, etc. We call these collections of sets *judgement classes*.

By ignoring the existence of judgement classes, we run the risk of serious ineffiencies in terms of the size and the speed of the classification tree: While at least some judgement classes (counterparty, transaction type, etc.) are very likely to be relevant, one way or another, for a consolidation point, any particular judgement is not. The result would be a classification tree that is both very big, and very deep (meaning slow, since it would long paths of the form: 'counterparty Goldman-Sachs?, counterparty J.P. Morgan?, counterparty Merill-Lynch?, etc. Further, these paths could be thousands of judgements long) which assign transactions in time linear in the number of judgements in the class)[11]

We obviously want to avoid this happening. Thus, instead of having trees split on binary judgments, we have trees split according to judgment

[11]These paths will be interleaved between judgment classes, of course, but they will be there.

classes, e.g.,

$$\begin{tikzcd} & \text{counterparty (CP)}=X \arrow[dl, "\text{X=G-S}"'] \arrow[d, "\text{X=Morgan}"] \arrow[dr, "\text{X=M-L}"] & \\ \ldots CT(D^*_{\text{CP=G.S.}})\ldots & CT(D^*_{\text{CP=Morgan}}) & \ldots CT(D^*_{\text{CP=M-L}})\ldots \end{tikzcd}$$

We can do this with a association table T from particular judgements to classification trees. Such tables are, at worst, $O(\log N)$ where N is the number of distinct judgements in a class; i.e. they are extremely fast. For a given judgement class, say, counterparty $= X$, the domain of T is a subset of the counterparties δ (note that not every member of δ may be mentioned in an exposure point). Thus we have

$$T(x) = CT(\{\ (R/\text{CP=M-L}, n) \mid (R, n) \in D, (-\text{CP=M-L}) \notin R\ \})$$

However we must also take care of the 'none of the above' option; i.e. where none of the explicitly mentioned values x is relevant; thus we also have

$$T(\mathit{otherwise}) = CT(\{\ (R, n) \mid (R, n) \in D, \neg\exists x.(\text{CP=x}) \in R\ \})$$

The result of this optimisation is obviously a classification tree with multiple points at the beginning, and essentially boolean judgements as we move towards the leaves. Thus we get our performance bound, as stated above:

If we assume at most one split of each judgement class, then the time to reach the leaves is bounded by $O(\sum_i \log |J_i|)$ where J_i is the set of the instances of a judgement class that we actually use.

Formally, this bound is not quite correct, since it is possible that a judgement class need to be tested so we must be careful not to delete both instances. This is, in practice, rare, however, and in most cases however, judgments can be used only once; and in this case we can make an important further refinement of T, as follows:

4.2.1 Exclusive judgements

When we build these multiway branches in the tree, we have assumed that the judgements, though gathered together in a class, are still essentially mutually independent. But this is clearly not always, or even only rarely, the case. Thus, for instance, if the counterparty is in Japan, then he clearly cannot be in the USA.[12]

To be precise, if a judgement class $A_1, \ldots, A_n$ (e.g. counterparty in the US, counterparty in France, counterparty in Germany, etc.) represent a

[12]An example where judgements are not necessarily exclusive is industries: a counterparty might be exposed to the auto industry, and to the finance industry.

disjoint partition if the set of transactions, and we partition the set of consolidation points on it, then clearly not only can we delete A_i from each entry assigned to that (as described above) but we can also ignore side any consolidation point which depends on the truth of A_j where $i \neq j$.[13]

Selecting the right partition

Notice that we do not specify which judgement to choose to partition the set of consolidation points at each step. We suggest choosing at each step that judgement class A_i which minimises $\text{Max} \bigcup_i \{|D^*_{A_i}|\} \cup \{|D^*_{otherwise}|\}$. This ensures that the longest possible path in the classification tree is as short as possible.

Deal assignment using the classification tree

With $CT(D^*)$ in hand, it is now easy to assign transactions to sets of consolidation points: starting at the root of the tree, we compare the the transaction against each judgement class $A(x)$ that we meet as we descend. The leaf of the tree that we eventually reach this way contains the names of all the consolidation points to which the transaction belongs.

5 Complexity, performance and optimisation

Essentially, in constructing $CT(D^*)$, all we have done is to ensure that we never have to ask the same question of the transaction twice, and tried to find the most useful question to ask at any point. Further, since the partition is on a judgment class at each step, the depth, and therefore the time to classify the transaction is more or less proportional to the number of judgement classes. Further, since all we are doing is negotiating a simply structured decision tree, possible performance of this algorithm is extremely high (back of an envelope estimates suggest that it should be able to classify a transaction against a typical real set of consolidation points in a few hundred thousandths of a second (as a result of some of the optimisations below; it may also be be necessary to consult one or more subsidiary tables (see §5.1), but access for these is also at worst log time).

The risk is that the payment for this speed can be enormous in terms of space, if we are not careful.

Ideally, each partition would be perfect, with the result that the branching set of possible consolidation points reduces radically at each step and the size of the tree is directly proportional to the number of consolidation points. Our world is far from ideal: it would be extremely unusual that some judgement class performs a perfect partition, but we should try to get

[13]This observation also provides a useful sanity check for any consolidation point definition, since no such can have, as one of its conjuncts, both A_i and A_j.

as close as possible. In fact the partition candidate we have proposed, although not explicitly specified to do so, should behave well by this criterion. Nevertheless, if we are to keep the size of the tree under control, we should also implement some optimisations which we now consider. They are more or less general, but all take address the nature of the specific structure of a set of consolidation points, as compared to a random set of consolidation points of the same size.

5.1 Factoring out a fixed class of points

In §4.2 we optimised the classification tree to take acount of judgement classes, rather than just single judgments; we now add a further optimisation to remove altogether from the tree any particularly large sets of consolidation points which can be assigned by some fixed collection of judgements.

A serious problem for the tree size is that a set of consolidation points is likely to have a large number of standard ones that are do not require the machinery we have developed here. For instance there may be a limit placed on each specific counterparty (of which there may be, let us say, a thousand—there are probably more). The effect of the existence of this subset of consolidation points D' can be best imagined if we first construct the classification tree without D', $CT(D - D')$, and then add D' separately. The result (as already observed) is a chain of 1000 judgements appended to each leaf (is the counterparty Merrill-Lynch?, is the counterparty Citibank New York? …, etc.) The result is both an enormous expansion in the size of the tree, and an enormous increase in response time (the effect is even worse if we have more than one of these sets).[14]

But before we enter the extension to the classification tree produced by D', we already know almost all the relevant consolidation points; all that adding the set D' does is to add an extra (very inefficent) lookup table to the leaf to find the one extra point that we need for this particular case.

Our optimisation is simply to detect when this is likely to occur, and do the lookup ourselves. Every exposure point in D^* has a *type signature*; which is a collection of the names of the judgement classes which have to be made in the process of deciding whether a transaction belongs to the consolidation point. Note that occasionally (if the judgements in a class are not exclusive—i.e. do not define disjoint sets of transactions) then the type signature should record the number of times a judgment class is invoked. It is an easy matter to count the number of instances of each different type signature in D^*, and then to identify any disproportionately large set of identically structured consolidation points. These can then be removed from the classification tree to a separate lookup table (a possibly multidimensional

[14]Note that this blowup is not detectable in the process of generating the normal form of D, since $D' = (D')^*$.

generalisation of that discussed in §4.2) which we can check for after exiting the classification tree.

It may make sense to extract more than one such set of consolidation points: for instance a system might, as a matter of course, set limits on both *company* × *instrument category* and *company group* × *instrument category*, where both sets would be very large relative to the total number of consolidation points.

5.2 Standard disjunctions

A further possible cause of classification tree blowup is the use of defined disjunctions in definitions. Thus, for instance, if transactions are naturally identified with cities, rather than countries (e.g. direct counterparty in Paris, direct counterparty in Lyon), it makes (as we have already observed) theoretical sense to define countries simply as disjunctions of their relevant cities; e.g.:

$$\begin{aligned}&\textit{Counterparty location France} \equiv\\&\qquad \textit{Counterparty location Paris} \cup \textit{Counterparty location Lyon} \cup \ldots\end{aligned}$$

Such an approach may be theoretically elegant, but it can lead to problems with large classification trees. The problem is that there may be many classification points which refer to, e.g.,

$$\textit{Counterparty location France} \cap P$$

which is expanded into a series of n conjuncts

$$(\textit{Counterparty location Paris} \cap P) \cup (\textit{Counterparty location Lyon} \cap P) \cup \ldots$$

each of which represents a different path through the tree. The roblem gets amplified with each level of definition (the next step up would then be to the EU, which would involve dozens of different cities).

The solution to this problem (if it has been identified as one) is simply to introduce a definition mechanism where such disjuncts are redefined as simple consolidation points; i.e. we introduce a new simple consolidation point into the system, of type, say, *country* so that *France* itself becomes an atomic judgment, and thus is no-longer expanded during the reuduction of consolidation points to normal form.

Alternative choices for the partition judgement

On the other hand it is extremely difficult to suggest an alternative choice for the partition judgement.

One possible alternative might result if we wanted to to make the classification tree as small as possible (i.e. select A to minimise $(\sum_i |D^*_{A_i}|) +$

$|D^*_{A_{otherwise}}|$) however, this should be done only after the other suggestions for reducing the tree size have already been implemented, since it runs the risk that we end up with a very suboptimal tree which first asks a lot of questions about small groups of rarely used consolidation points, leaving the most common points at the end of relatively long paths in the tree.

6 The problem of time

The most serious problem with the boolean language of consolidation points we have presented is that the model does not treat time. It seems to provide us with no way to define, say, the set of 'short-term' transactions (where 'short-term' is usually interpreted as meaning 'matures inside one year'). In this section we consider the extensions that are needed to make use of temporal concepts, and argue that these extensions are purely 'technical'; i.e. they fit naturally within the conceptual framework of boolean algebra that we have already developed.

First, we note an important feature about the sorts of temporal judgements we want to make about transactions: they are *relative*, not absolute; i.e. we are not usually interested in classifying transactions with respect to time points such as *June, 2003*, but rather with respect to *in six months time*. Second, not only are the judgements we want to make relative, but there is clearly only a small fixed set of them; e.g. we might need

- *within the last week*,
- *within the next week*,
- *within a month*,
- *short-term* (*within a year*),
- *long-term*,
- *within five years*,
- *within ten years*

etc. But these judgements simply define new simply consolidation points, which can be mixed with those that we already have.

In fact temporal judgements require almost no new machinery. We can build the classification tree treating temporal judgments as essentially no different from other derived judgements: the deal arrives in the system stamped with (one or more) absolute time values, and the relative judgements we need are simply derived from these. We only have to realise that those deals for which some relative judgement may change overnight (e.g. a transaction could change from long to short-term) have to be redistributed. But given the speed of the classification algorithm, it would not be a problem, if necessary, simply to reclassify all transactions sometime during close of business (although some more sophisticated method which calculated merely the *delta* on the various consolidation points could be easily implemented, and would have clear advantages).

Remarks on temporal logic

We have argued that we can get all the temporal facilities that we need simply by defining a few extra simple consolidation points. However this is not the only possible approach. An interesting and (at least theoretically) natural alternative to consider might be, rather than to extend the set of consolidation points, to extend the set of operators on consoliation points; i.e. to add temporal connectives, extending boolean algebra to a modal temporal logic. This would give us instead of a fixed set of new conosolidation points, a fixed set of new connectives, which could then be applied to any already defined set.

Such an approach would clearly be much more flexible: fortunately it is not clear where we would need this flexibility—it would only be necessary if the structure of a transaction changed radically over time, but in practice, if ,e.g., the counterparty is currently Merill-Lynch, then it will still be Merill-Lynch in a week. This is forunate, since the structure of temporal logic is a great deal more complex, with the result that there is no similarly effective way of classifying transactions against temporal propositions.

7 Modifying the set of consolidation points

A further, and important, advantage beyond speed and expressiveness with the approach we describe here, is that the core data structures easily absorb minor modifications without serious effects on performance (further, these modifications can be reasonably performed even on a running system). This means that, e.g., risk managers are able to add new consolidation points, or even modify those already in the system, without automatically triggering a complete recompilation. For instance OPEC oil producers might suddenly become interesting, with the result that it is suddenly necessary to add a collection of new consolidation points to classify and monitor transactions related to that group. Or it might even be necessary to change the definition of OPEC itself (let us imagine that the United Kingdom decided to join), in which case all consolidation points referring to OPEC would need to be adjusted.

7.1 Adding a new consolidation point

Adding a new consolidation point to the classification tree requires only a series of 'local' patches. All we do is to convert the consolidation point, as usual, into normal form, and then for each conjunct of the disjunction, move down the three from the top, comparing the judgements at each branch with those in the conjunct:

- If the judgement is not relevant, then we take all branches,
- on the other hand, if the judgement is relevant, we take the appropriate branch, and delete the judgement from the conjunction.

Finally, on reaching the leaves, either every judgement in the conjunct has been deleted, in which case we add the name of the consolidation point to the leaf set, or there are some judgements left, in which case we extend the tree in the obvious manner.

The extended tree is almost unchanged, the only modifications are at those leaves which transactions that might belong to the new consolidation point reach, thus while the tree is no-longer quite optimal it is very nearly so, and the only suboptimal behaviour is with respect to these transactions (where a few extra judgements may have to be made). Clearly, the classification tree should have no problem absorbing almost any reasonable number of such extensions without serious performance degradation (especially if the number of consolidation points used to build the original classification tree is large, since the the result will then be almost optimal for most extensions).

Note that when we add a new consolidation point, we do not automatically identify the transactions already in the system which correspond to it. However we have already observed that it is an easy matter to construct this set.

7.2 Deleting consolidation points

Consolidation points can be deleted just as easily as they can be added, using the same procedure.

7.3 Modifying consolidation points

More complex is the case where it may be necessary to modify one or more consolidation points (e.g. if the definition of OPEC were modified to include the UK, then presumably a large number of separate consolidation points would be affected).

However the basic principle is similar to that for adding/deleting a new consolidation point. We could simply delete all the old definitions, and add the new ones, but that would be more computationally intensive than necessary. Instead, we can simply calculate the *delta* for the modification, which is likely to be much smaller, and implement that: First must find the set of affected consolidation points, then we must identify:

1. the set of leaves of the current classification tree to which the old versions consolidation points are assigned
2. the leaves of the (possibly extended) tree to which the new versions of the consolidation points are assigned

Finally, we can calculate the difference (i.e. the delta), and adjust those leaves which have changed as a result (i.e. if a point has been deleted or added to some leaf, or if it is added to a leaf of a new extension of the classification tree.)

8 Reservations

It is not clear that it is possible to design a 'perfect' reservation logic. The various requirements are simply not reconcilable. In this section we discuss some of the problems, provide a formal analysis, and presenta basic method for calculating the resrvation/limit pairs which may interfere with one another.

8.1 The non-local effects of reservations

The core problem with reservations is that they are not 'local': a transaction which in itself has no relation to the consolidation point to which a reservation has been attached may nevertheless influence it.

Consider an example: limits of 10 units have been placed on the consolidation points for

1. *South America*, and
2. *banks*

in our EMS, and we would like to guarantee that trader X has access to 5 units for future transactions against *South America* $\cap$ *banks.* While there is currently no limit on that consolidation point, nevertheless, the limits at 1 and 2 are clearly relevant, implying, at the least, that we are not able to reserve more than 10 units. However, since there are currently no reservations booked in the system, we have no problem doing that.

Now consider that a limit of 10 units is placed against a third reservation point, say *Banks* $\cup$ *Oil industry*, and trader Y immediately tries to book a transaction of 10 units against *British Petroleum.* Clearly, an immediate consequence of accepting this transaction would be to void X's reservation: as a result of the third limit, the system will not allow any more transactions to be booked against banks and, therefore, not against South American banks.

8.2 A refinement of the problem

The problem cannot be resolved simply by having the system refuse to accept the transaction, as we can see if we refine the example slightly, by refining consolidation point 2 to *banks* $\cup$ *Car manufacturers*. Now Y's transaction does not void X's reservation; it merely means that X can book transactions only against car manufacturers.

This refinement also raises a corollary question: what exactly does it mean to make a reservation of U against $A \cup B$: that we can book transactions of total value U against this consolidation point, or that we should be able to book all of U against A or against B? The later option seems clearly unacceptable (the result would be, e.g., that the system should ensure that X is always able to book the whole reservation against a single transaction with the smallest bank in the Lima suburbs, which is clearly not what is intended—and also, as a side effect, lock down all other business with South American banks). On the other hand, if we accept the first option, then X may find that he has a reservation, but that it is not usable, since all most interesting bits have been 'chipped away' by other transactions (local or non-local) leaving him to do business with suburban banks in Lima after all.

8.3 Formal analysis and notes on implementation

The practical details of the implementation of reservations is clearly complex and application specific, so we will restrict ourselves to generally applicable principle of how to detect when a limit and a reservation interact.

Consider a reservation against a consolidation point A, and a limit against a consoliation point B, where $A \cap B \neq \emptyset$. If a transaction t falls into $A \cap B$, then it is relevant since it is also booked against A, but even if it falls into B/A, it still has, as explained, an effect on the reservation: if it uses up the limit on B, then the reservation against A is only bookable against A/B (the 'small banks in Lima' set, which at the very least, we now have to make sure is still allowed).

The problem then is to calculate those consolidation points which carry limits relevant for a reservation consolidation point. Fortunately, we have most of the machinery already available. Both limit points L and reservations R are associated with consolidation points and thus are equivalent to formulae:

$$\bigcup_{i} \bigcap_{1 \leq j \leq m_i} ([-]L_j^i)$$

$$\bigcup_{p} \bigcap_{1 \leq q \leq n_p} ([-]R_p^i)$$

Further, the two sets intersect precisely if there exists a pair of a conjunct in R and a conjunct in L which share a non-empty common subset of simple consolidation points, and the signs on the points in that common subset match.

Fortunately, we already have a list of all existing limit consolidation points, reduced not only to normal form, but factored into conjuncts, D^*. All we have to do now is to reduce the reservation (or set of reservations) to similar form, and perform a join according to the method for identifying an intersection we have just given.

Note that in calculating the join, we should also take account of the existence of exclusive judgements, as discussed in §4.2.1: if two conjunctions share a positive judgement type then, even though the judgments themsselves are different, the two conjunctions do not intersect.

8.4 Remarks on reservations and limit management

There is clearly scope for a variety of different reservation facilities which attempt to alleviate the problems we have described in different *ad hoc* ways, but we make two remarks in passing: first that there is no requirement that a reservation be for a particular group of people: is it simply a reservation against any consolidation point. Second, that a facility to set a simple *cumulative* reservation, to which transactions are automatically booked, against a consolidation point is a powerful tool for limit management and thus should be made available alongside any other facility. (For discussion of this, see §9.)

9 Limit management

In this section we provide an example of automatic limit management using the facilities we have defined.

9.1 Manual limit management

The scenario is as follows: a risk manager has 15 units of resources which he can distribute between three trading areas, the high risk/high return A and the medium risk/medium return B, and the low risk/low return C; he decides to allocate, by default, 5 units to each area. On request he is willing to reallocate some resources temporarily, if they happen to be free, from the higher risk to lower risk areas but not *vice versa* (i.e. from A to B or C, and from B to C, but not from C to B or A, or from B to A). However he is also loath to reallocate all resources from even A (let us say that he will never reduce the allocation to any area below 3 units, even the resources are not currently in use).

A regime like this clearly requires constant human monitoring. Not only does the risk manager have to reallocate resources by hand on request, but even after he has done so, he must monitor the three areas so that as soon as resources are released, they can be redistributed to bring the the allocations back to their defaults as soon as possible.

9.2 Automatic limit management

How might a risk manager use the facilities we have described to automate his regime?

9.2.1 The limit structure

Rather than assign limits to individual transaction types, the risk manger creates, and limits, the following consolidation points:

1. $A \cup B \cup C$, limited to 15 units,
2. $A \cup B$ limited to 10 units,
3. A limited to 5 units

Note that there is no limit at all on transactions of type C alone, these are constrained only by the 15 unit limit on $A \cup B \cup C$; similarly for B which is constrained only by the limits on $A \cup B$ and $A \cup B \cup C$.

An examination of this arrangement shows that indeed—at least if we think in terms of the manual model—resources can flow from high risk to low risk areas and back again without manual intervention.

9.2.2 The reservation structure

However it is still possible for, e.g., a trader to book a 15 unit transaction against C, thus using up all the resources that should be available to A and B, even though we have decided that there shall always be at least 3 units available in each of those areas.

The solution is as follows: we supplement the limits we have assigned to the various points with a pair of reservations of 3 units against A and B. Note that these are not reservations in the name of some person; they are simply reservations for transactions of types A and B. Now, even if nothing is currently booked against A or B, the largest transaction that can be booked against C is 9 units; i.e. the other half of the constraint that the risk manager defined on resource reallocation has been implemented.

If we examine this combination of reservations and limits, then clearly it satisfies all the requirements of the original regime, except that now it is completely automated; the risk manager need never intervene manually.

9.3 Emergency shutdowns

A second important form of manual limit management is ‘emergency shutdowns’: in an emergency a risk manger might want to suspend temporarily all transactions which increase exposure in some area. This area might be, say, all companies registered in Gulf states (rumour of war), or all subsidiaries of a Japanese bank (rumour of insolvency).

The sets of transactions which we want to block are definable as consolidation points, therefore all we have to do to suspend all such business (and, as importantly, if necessary, to free up again as quickly) by setting a limit against a single consolidation point of 0 units. This limit could be a consolidation point already defined in the system (e.g. *Company group=‘IBM’*), but it could equally easily be something like OPEC countries other than Kuwait with a poor longterm rating:

$$\begin{aligned}&\mathit{Counterparty/OPEC}\\ &\cap -\mathit{Counterparty\ location} = \mathit{Kuwait}\\ &\cap \mathit{Counterparty/Country/rating\ longterm} \leq A.\end{aligned}$$

10 Exceptions to the model

There are always exceptions to a formal model, and what we describe here, though fast, expressive, and flexible, is no different in that respect: there exist some more or less natural classes of consolidation point that are not naturally captured by it, and for such cases it will still be necessary to add some exception logic. However at least in the cases we have encountered or imagined, the necessary facilities are not complex.

Consider, for example, the case of an issue limit. Some counterparty C issues a series of bonds. These bonds are identical in every way, except for the issue number, which is unknown in advance. The risk manger wants to monitor exposure on these bonds, and, alongside limiting exposure to the set of all such bonds as a set, to limit exposure *on a per issue basis.* We can easily define a consolidation point CP which defines the set of all these bonds together, but we have no way to formulate the requirement that there should be a limit on each issue of this type. In such a case, we need to be able to attach logic to CP which then sets up, automatically, a consolidation point and a limit. Such an arrangement is exactly like the machinery we proposed for automatic maintenance of, e.g., country-risk consolidation points, except that it is triggered by a transaction landing in a particular consolidation point. The same technique can be adapted to any particular special case where the logic as provided is not suitable.

A further useful hook for handling exceptional judgements is the general optimisation method described in §5.2 for treating definitions which threaten to make the classification tree too large; this can also be used to integrate cleanly arbitrary predicates that do not fit directly into the model developed here (in fact we have already used it to implement temporal judgments, in §6).

11 Comparison with other solutions, summary and conclusions

In summary, we have presented a logic/technology for the modelling and implementation of a system for monitoring and limiting counterparty exposure, and in particular for performing pre-deal limit checks, in real time. Our architecture has the following properties:

- It provides an expressive and intuitive language for defining consolidation points, which allows a direct translation from original natural language specifications. This language has been checked against the consolidation point structure of two different large German banks.
- Transactions can be classified for pre-deal checks against limits essentially instantaneously (we estimate in hundreds of a thousandth of a second, or approx. $O(\sum_i \log N_i)$ where N_i are the sizes of (the utilised parts of) the judgment classes). This estimate is dependent on the particular properties of sets of consolidation points. Further, performance should be essentially independent of the scale of the system (as a result of the log bound).
- It allows a user to modify not just limits, but also introduce arbitrary new consolidation points to the consolidation points relevant for pre-deal limit checks, in real time and have these limits take immediate effect (thus allowing a risk manager, e.g., to suspend immediately any definable class of transactions).
- It allows a risk manager to implement a flexible limit management regime automatically.
- We provide a general analysis of reservations, and a method for identifying in general when limits and reservations interact.
- We also define clean techniques for extensions, when necessary, to deal with circumstances which the model cannot treat.

It is important to emphasize that in order to be practical, we require certain optimisations, which reduce drastically the otherwise exorbitant memory requirements of our architecture. Since the other solutions for this problem are offered as commercial software, it is difficult to make explicit comparisons, since we have not been able to discover detailed information about underlying principles (extensional or intensional). Comparisons must thus bbe on the basis of user manuals when available, presentations, and anecdotes. However we have seen no solid evidence that existing systems offer the sort of flexibility described here. Presentations we have seen tend to present the space of transactions in intuitive terms as a ‘cube’, with consolidation points defined as consisting of one or more subcubes of this space; a model like this corresponds to the fragment of our Boolean language without the negation operator, but there is little or no suggestion of the full language as we have defined it. No mention either is made in commercial presentations of what we see as important properties of our model: first that it is insensitive to scale; further, commercial systems seem to have detectable performance limits defined as a function of the number of consolidation points. Finally, existing commercial presentations do not discuss the possibility of entering new consolidation points into the pre-deal checking logic, and therefore of suspending an arbitrary class of transactions. (Note that this is different from the much less strenuous requirement that a sytem be able to generate reports on the exposure associated with an arbitrary consolidation point).

We should add that, alongside commercially available systems, we have also seen an interesting design by Philipp Brune and Martin Exner, which implements a logic similar, on inspection, to what we develop in this paper. However this design has been developed under the constraint that it be implemented in terms of a relational database, with the result that predeal limit checks are $O(N)$ in the number of (disjuncts of normalised) consolidation points, with a large constant factor (the result of the overhead of the relational logic), and that it seems to be less flexible than our design in, e.g., the freedom to modify the pre-deal checks in a running system.

Acknowledgements

The author would like to thank Philipp Brune, Martin Exner, Peter Friedl and Robert Pfund, Mireille Khazaka, Karsten Wilhelmsen, and in particular Jens Kramer of Remark Consulting, for discussions which in various ways helped with the development of the ideas in this paper.

Appendix: a note on a possible natural modal extension for treating derivative transactions

We have discussed, and dismissed, the idea of modal extensions to our language for the purposes of formalising time judgements, on the grounds that we neither need the expressiveness, nor know how to implement the resulting system efficiently. However it is interesting to record briefly—if only for the sake of theoretical interest—another modal extension that might be useful, efficient to implement, and consistent with the model we have developed in this paper.

In a more complex counterparty monitoring system, we are sure to encounter derivative transactions. In most of these, the immediate counterparty is of interest, however in the special, but common, case of a credit derivative, we are interested in the properties of both the direct counterparty and the underlying counterparty. But the mapping from set of instruments to set of underlying instruments has obvious similarities to an accessibility relation in a Kripke-style model. Further, the implementation we have describe in this paper could be easily extended to take account of it (since the the 'modality' has a particularly simple meaning: just a redirection to the underlying instrument).

Thus in theory there is no reason why we could not extend our language of consolidation points (viewed as a propositional logic, rather than a boolean algebra) with a modal connective for handling derivative transactions. Indeed, taking the common (technical) view of modal logic as simply the controlled introduction of, essentially, predicate logic into a propositional setting, then this would be an example of the controlled introduction of relational algebraic concepts into a boolean setting.

It should be added that in practice, so far, we have not seen the need for this sort of flexibility: the same approach as we have proposed with time—of adding a few specific new judgements—seems to be sufficient.