DESIGNING OBJECT-ORIENTED REPRESENTATIONS FOR REASONING FROM FIRST-PRINCIPLES

by

Michel Benaroch
Information Systems Department
Leonard N. Stern School of Business
New York University
40 West 4th Street
New York, New York 10003

July 1991

Center for Research on Information Systems
Information Systems Department
Leonard N. Stern School of Business
New York University

Working Paper Series

STERN IS-91-19

Abstract

Modeling expert knowledge using "situation-action" rules is not always feasible in knowledge intensive domains involving volatile knowledge (e.g., trading). The explosive search space involved in such domains and its dynamic nature make it extremely difficult to setup a rule base and keep it accurate. An alternative approach suggests that in some domains many of the rules expert use can be derived by reasoning from "first-principles". That approach entails modeling experts' deep knowledge, and emulating reasoning processes with deep knowledge that allow experts to derive many of the rules they use and justify them. This paper discusses the design and implementation of an object-oriented representation for the deep knowledge traders utilize in a business domain called hedging, which is knowledge intensive and involves volatile knowledge. It illustrates how deep knowledge modeled using that representation is used to support reasoning from first-principles. The paper also analyzes features of that representation that we have found to be extremely beneficial in the development of a knowledge-based system called INTELLIGENT-HEDGER. Based on our experience we feel that, with minor modifications, this representation can be used in other managerial domains involving financial reasoning.

Contents

1 Introduction

Financial hedging is concerned with the design of hedge vehicles that protect against losses due to uncontrollable events. A hedge vehicle involves the purchase and/or sale of financial securities such as bonds, stocks, and options. It allows a trader to control the balance between risk and reward. This balance is a function of a trader's prediction of economic variables' future behavior and its effect on the market value of securities.

Hedge design can be conceptually formulated as a multi-objective optimization problem that can be shown to be NP-complete. This problem involves several complexities. The two major ones are the need for a number of constraints that can be only specified qualitatively, and more importantly the explosive search space whose elements (i.e., hedge vehicles) are constantly changing due to the increasing globalization and dynamics of today's capital markets. These complexities make the problem extremely difficult to formulate and solve using quantitative techniques.

Benaroch and Dhar (1991b) suggest that hedge design could be solved using a knowledge-based approach. These authors argue that a knowledg-based approach that uses situation-action rules to model the knowledge of hedging traders is not likely to work exactly because of the two above complexities (Benaroch & Dhar, 1991a). They also propose that in order for a knowledg-based system to work well in hedge design, it should reason greatly based on the way traders' reason in hedge design.

Experienced traders deal with the above complexities by reasoning mainly based on "first-principles". In such reasoning traders derive much of the knowledge needed in hedge design by making infrences based on deep knowledge about atomic relationships between objects in the domain (i.e., relationships between fundamental economic variables and financial securities). This kind of reasoning allows traders to make good decisions even in situations they have never encountered, that is, new situations that are constantly being created by the dynamics of capital markets. It also allows them to justify most of their decisions intuitively. Traders also reason qualitatively with abstracted knowledge about types of similar securities, rather than about each security separately. This kind of reasoning helps them avoid much of of the complexity involved in reasoning about certain parts of the design process.

This paper discusses the object-oriented representation used in a knowledge-based system called INTELLIGENT-HEDGER that we have developed to assist hedging traders (Benaroch & Dhar, 1991b). This representation is designed to accommodate special knowledge requirements of the domain, and to facilitate reproduction of the kinds of reasoning traders use in hedging. It provides a number of important features. One is the ability to capture traders' deep knowledge about hedging, and to support direct reference to the basic objects of the domain in reasoning from first-principles. This allows for the derivation of many situation-action rules one might elicit from traders, if one were trying to develop a rule-based system. A second feature facilitates abstraction of knowledge about financial securities and economic variables to support reasoning qualitatively about parts of the design. The third feature is the use of inheritance properties via specialization relationships, something that can reduce the amount of knowledge one needs to elicit and store in a knowledge base. Finally, the representation organizes knowledge in a modular fashion to provide maximum flexibility and to minimize the amount of effort needed to keep a knowledge base current.

The rest of this paper is organized as follows. Section 2 first explains the purpose of hedging using two examples, and then illustrates the knowledge and computation complexities involved in one problem that deals with the configuration of hedge vehicles early in the design process. This problem is complex and requires reasoning from first-principles. Section 3 discusses the design of the object-oriented representation we have used in INTELLIGENT-HEDGER, and its implementation in C++. Sections 4 and 5 explain why features of this representation make the configuration of hedge vehicles feasible, and demonstrate how these features are used in the configuration part of the design. Section 6 concludes with the cons and pros of the use of our object-oriented representation in hedge design.

2 Hedge Design Complexities

A *hedge vehicle* can be defined as the purchase and/or sale of financial securities. We distinguish between two types of hedge vehicles — *generic* and *compounded.* A *generic hedge vehicle* involves selling or buying securities of one type, while a *compounded hedge vehicle* involves selling and/or buying securities of more than one type.

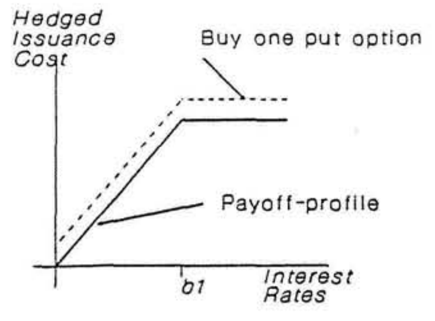
Like many other design problems, hedging can be viewed as a two-phase process. In the first phase, the *configuration* phase (Williams, 1990), the space of all hedge vehicles is searched to identify vehicles that satisfy some feasibility constraints. In the second phase, the *refinement* phase (Mittal & Araya, 1986), design parameters associated with feasible hedge vehicles (e.g., liquidity and setup cost) are constrained to identify one hedge vehicle that satisfies best some optimality constraints.

Conceptually, the configuration phase can be viewed as involving a series of subproblems, each entailing a search for all vehicles that satisfy only one feasibility constraint. Since a subproblem for any given constraint can be solved independently from the other subproblem, it requires its own specific knowledge items about elements in the search space and its own specific reasoning process to determine which elements satisfy that constraint. The rest of this paper concentrates on the solution of one feasibility constraint in the configuration part of the design process. It is used to illustrate few of the knowledge modeling and computation complexities involved in hedge design, and to explain how the use of a customized object-oriented representation can avoid these complexities.

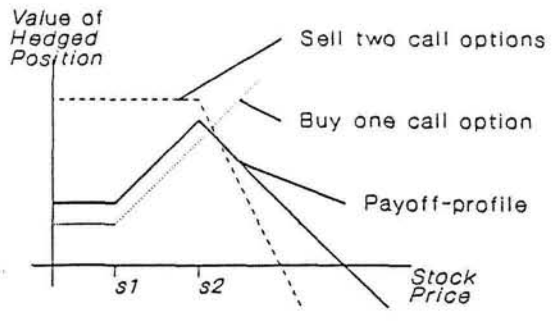
2.1 Configuring by the Payoff-Profile Constraint

One major feasibility constraint used in the configuration of hedge vehicles is the *payoff-profile* constraint. A *payoff-profile* is the "internal" option strategy a trader underwrites to specify what he is willing to pay and risk based on his assessments of the effects of the predicted future behavior of economic variables on the value of securities. To derive a payoff from such assessments, a trader invariably wants to use a hedge vehicle that provide the payoff-profile he specifies.

Consider for example a firm that plans to issues bonds to raise capital. The firm believes that interest rates are likely to increase prior to the issuance date. In such a case the firm will have to offer a higher rate on its bonds, something that will increase the issuance cost. The firm wants to protect itself against an increase, while preserving the ability to benefit from a decline, in interest rates. It therefore defines a "cap" payoff-profile (see Figure 1a). One generic hedge vehicle that provides a "cap" payoff-profile is the purchase of put options



(a) A "cap" payoff-profile



(b) A "ratio-spread" payoff-profile

Figure 1: Two payoff-profiles

on some bond B with exercise price b_1.[1] An increase in interest rates will cause the price of B to decline below b_1, which will allow the firm to sell bonds at the higher price b_1 and make a profit to offset the extra cost of issuing bonds at a higher rate. Alternately, a decline in interest rates will make the put valueless, but will allow the firm to issue bonds at a lower rate and make a profit to offset, and more, the cost paid for the put.

Traders often define complex payoff-profiles that are provided only by compounded hedge vehicles. For example, if a trader believes that the price of stock S will increase above s_1, but not above s_2, that trader can define a "ratio-spread" payoff-profile (see Figure 1b). One compounded hedge vehicle that provides that payoff-profile is called a ratio-spread. It involves two generic vehicles. One is the purchase of a call option on S with exercise price s_1,[2] which in case of a price movement above s_1 will allow to profit from buying stocks at the lower price s_1 to offset, and more, the cost paid for that call. The other vehicle is the sale of two calls on S with exercise price s_2, which allows to profit from the cost paid for the calls sold to another party that believes the price of S will move above s_2. Note that a ratio-spread payoff-profile is in effect a synthesis of the payoff-profiles provided by the two above generic vehicles.

These examples show that early in the design process traders specify a payoff-profile qualitatively as a two-dimensional piecewise linear function. A payoff-profile can be symbolically expressed as $\frac{dVHP}{dHV} = \begin{cases} qdir_1 & HV \in qval_1 \\ \ldots & \ldots \\ qdir_n & HV \in qval_n \end{cases}$, where HV is the variable being hedged (e.g., interest rates), VHP is the value of a trader's hedged position, $qdir_i \in \{-1, 0, 1\}$ (i.e., {decreasing, steady, increasing}) is the qualitative direction of change of VHP over $qval_i$, a qualitative range of values of HV.

2.2 Problem Definition

One problem hedging traders face can be summarized as follows. Given a goal payoff-profile that is specified qualitatively by a trader, configure all hedge vehicles that provide that goal payoff-profile. Configuring one hedge vehicle entails identifying types of securities and associating each one of them with a buy or a sell action, ordering their exercise prices,

[1] A put gives its buyer the right to sell, and obligates its seller to buy, the underlying security at an agreed exercise price at some future expiration date.

[2] A call gives its buyer the right to buy, and obligates its seller to sell, the underlying security at an agreed exercise price at some future expiration date.

and indicating the number of securities necessary from each identified type.

This configuration problem can be formulated as a process of searching the space of all payoff-profiles of every possible hedge vehicle against a goal payoff-profile. However, the search space involved in this problem is explosive. A generic hedge vehicle may provide a different payoff-profile under a different market situation (i.e., a different combination of values of economic variables). Furthermore, while the number of generic vehicles is large, the number of compounded vehicles (i.e., permutations of generic vehicles) is theoretically infinite. Thus, it is not feasible to prestore all payoff-profiles of every vehicle for selection, especially since new securities are constantly introduced to the market and matured ones are eliminated.

The previous examples demonstrate how traders handle the combinatorial nature of the configuration problem. They construct configurations whenever necessary by reasoning from first-principles. They construct a generic vehicle by deriving its payoff-profile using a qualitative causal analysis of atomic relationships between objects in the domain, and a compounded vehicle by synthesizing its payoff-profile from payoff-profiles of generic vehicles. In addition, they reason qualitatively with abstracted knowledge. This allows them to analyze whole classes of vehicles (e.g., a call on security S with exercise price s_1), rather than each specific vehicle separately (e.g., a January call on 100 IBM shares with $45 exercise price).

Constructing configurations of hedge vehicles, rather than prestoring them for selection, entails the use of a representation that can capture the deep knowledge traders use, and facilitates the emulation of first-principles reasoning processes traders use to construct configurations of hedge vehicles. In the following section we explain the principles that guided the design and implementation of one such representation.

3 An Object-Centered Representation

Our analysis of guides to hedging (e.g., Baecher & Goodman, 1988; Sutton, 1988) suggests that much of the deep knowledge traders use in hedge design centers around two types of structural entities and atomic relationships between them. These structural entities are financial securities and fundamental economic variables to which the value of securities is sensitive. A security can be generic (e.g., stock) or compounded (e.g., swap). A generic security can be either a basic one (e.g., bond) or a derivative one (e.g., option on bond). A compounded security is made from a number of generic securities. There are many thousands of securities that traders must look at while designing hedge vehicles.

We have identified two types of atomic relationships between structural entities. One is a specialization relationship between securities (e.g., T-bills, T-notes, and T-bonds are types of Treasury securities). Indeed, our analysis of the attributes used to describe securities shows many similarities among securities. The other type includes correlation and causal relationships between domain entities. The value of a basic security is determined by the behavior of fundamental variables via correlation relationships (e.g., bond prices goes up when interest rates go down). The value of a derivative security is, in addition, determined by the value of its underlying security via a causal relationship we term the *ON relationship* (e.g., call ON stocks). Correlation and causal relationships determine the payoff-profile a hedge vehicle will provide under any given market situation.

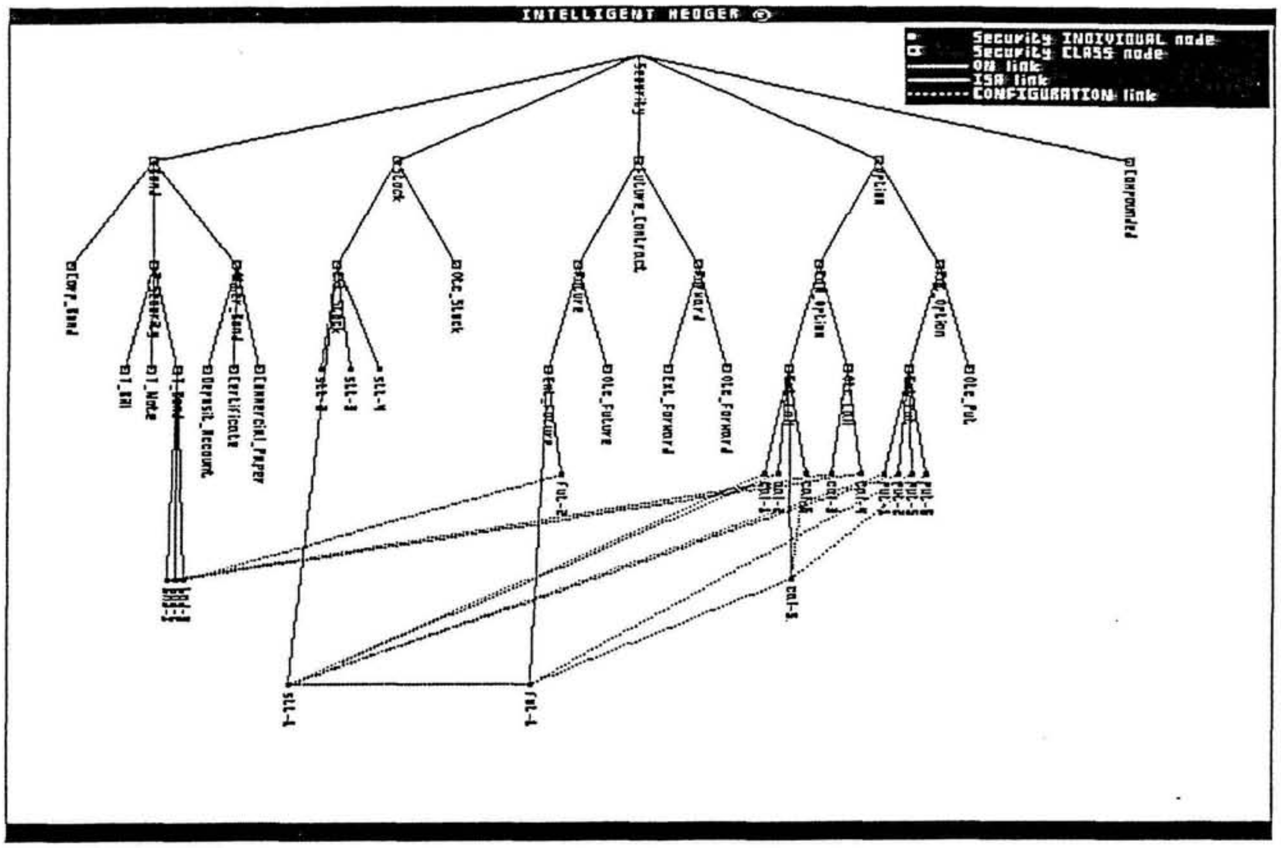


Figure 2: An ON-ISA semantic network

3.1 An Object-Oriented Semantic Network

We store knowledge about securities in an object-oriented semantic network to which we shall refer to as the *ON-ISA network* (see Figure 2). Since the number of compounded securities is very large and since knowledge about a compounded security can be derived from knowledge about the generic securities used to create it, we store in this network only knowledge about generic securities. Nodes in the ON-ISA network are organized in an ISA hierarchy to reflect the specialization relationship between securities. Each node in this network can represent either a class of generic securities or an instance of a generic security.

A node can be linked to other nodes by three types of links. The first is an ISA link that connects a node to another node representing one of its super classes. A node can be connected to more than one other node via ISA links to support multiple inheritance. The second type is called an ON link. It connects a node representing a derivative security to the node representing its underlying security. The third type is called a CONFIGURATION link. It connects a compounded security to one of the generic securities used to create it. This kind of link is useful in the derivation of knowledge about compounded securities. We shall illustrate the use of this type of link in Section 5.

Every node stores a large number of deep knowledge items that describe, or can be used to derive other knowledge that describe, attributes of the type of security it represents. The items of knowledge we choose to store in a node are determined by the kind of reasoning required with that node. For example, for every given feasibility constraint we identify the items of knowledge about a hedge vehicle that a trader needs to analyze to determine if that vehicle satisfies that given constraint. Let us explain what are the knowledge items we store for the payoff-profile constraint.

In every node representing a class of securities we store as an attribute the structural equations used to derive the *pricing model* of that class. A correlation/causal relationship between domain entities is modeled in Finance by a structural equation. The analytical solution of a set of structural equations describing the major relationships between the value of securities in a certain class and the variables to which it is sensitive (hereafter, structural model) is called the *pricing model* of that class (Elton & Gruber, 1988). A pricing model is used to derive the fair price of securities in the same class. The structural model used for a certain class of securities can be a specialization of the structural model for securities in a super class. For example, the structural model for bond options is a specialization of the Black-Scholse model, which is used to derive the pricing model for other option types.

We store structural equations for the pricing model of a class of securities rather than use physical links to represent correlation relationships for two reasons. Structural equations can be used with qualitative causal reasoning techniques to derive the payoff-profile provided by a generic hedge vehicle under various market situations (see Section 4). In addition, the pricing models that every trader has developed for himself over time can be directly plugged into the knowledge base. This can reduce the amount of effort needed for knowledge elicitation, especially since these models are subject to modifications.

The ON-ISA network has the characteristics that we have suggested are needed in a domain such as hedging. It captures trader's deep domain knowledge in the form of atomic relationships between structural domain entities. It also provides inheritance properties of the ISA hierarchy to facilitate abstraction of knowledge and to minimize the amount of stored knowledge. Finally, it uses object-oriented concepts to provide maximum modularity and flexibility, in support of the notion of an open KB architecture.

3.2 Network Implementation

We have implemented the ON-ISA network in C++. Recall that a node in the network can represent a class of securities or an instance of a security. Object oriented programming languages such as C++ provide inheritance only of attributes from a class to its derived classes (Rumbaugh *et. al.*, 1991). Defining in C++ a hierarchy of objects similar to the one in Figure 2 would therefore require storing redundantly the value of attributes that is identical for all securities in the same class (e.g., structural model). The kind of inheritance needed in the ON-ISA network is of attributes and their value. In other words, we want all nodes that are instances of a certain node, say C, to inherit via an ISA link the value of an attribute, say A, in node C, without having to store attribute A in every instance of C. This kind of inheritance reduces the need to store knowledge redundantly.

The hierarchy of object classes we have defined in C++ is presented in Figure 3. Note that this hierarchy is different from the one in Figure 2. The highest object class in this hierarchy is called `SECURITY`. It is described by attributes (i.e., data members) that are common to all securities, such as the security spot price and an array of pointers to an object of the same class that are each used as an ISA link. One derived class is called `DERIVATIVE-SECURITY`. It has an attribute that is a pointer to an object of class `SECURITY` that is used as an ON link. Another derived class is called `SECURITY-CLASS`. It is described by attributes such as the structural model of a class of securities and the transactions involved in setting up a security of that class as a hedge vehicle. All other subclasses are mainly derived from these three classes.

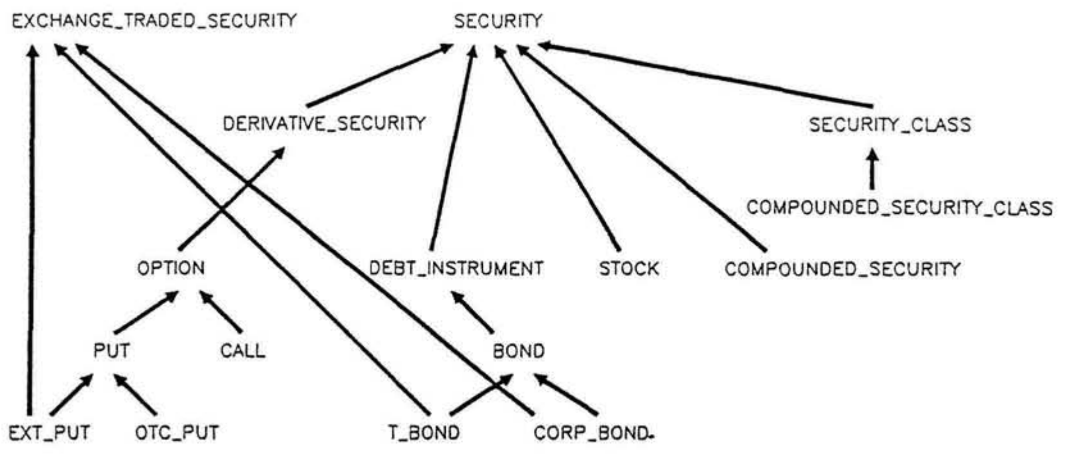


Figure 3: Part of the hierarchy of object classes defined in C++

3.3 Reasoning with the Network

Given a goal payoff-profile that is specified by a trader and given an ON-ISA network storing knowledge about the structural model of every class of generic securities, we need to construct every type of generic hedge vehicle that provide that goal payoff-profile. If no type of generic vehicles provides the goal payoff-profile, we need to construct every type of compounded vehicles that provides the goal payoff-profile.

Each C++ object has methods (i.e., function members) that allows it to reason with knowledge about itself and to communicate with other objects. We next describe two methods that can emulate the first-principles reasoning process traders use to configure hedge vehicles. One method called `Produce-Payoff-Profile` is of class `SECURITY-CLASS`. It is used to derive the payoff-profile of a hedge vehicle involving generic securities in a certain class using the structural model stored in the node representing that class. The other method called `Synthesize-Payoff-Profile` is of class `COMPOUNDED-SECURITY-CLASS`. It is used to synthesize compounded hedge vehicles from generic ones. In the following two sections we explain when these methods are invoked and how each one of them operates.

4 Constructing Generic Hedge Vehicles

Given an ON-ISA network storing knowledge about generic securities, it is necessary to construct every possible generic hedge vehicle, to produce the payoff-profile it provides under a trader's previewed market situation, and to compare the produced payoff-profile against the goal payoff-profile. The first example in Section 2 shows that the payoff-profile of generic vehicles of the same type can be produced by a qualitative causal analysis of the structural model of the class of generic securities involved in that vehicle. That analysis is actually a qualitative simulation that derives the value of a trader's hedged position under the specific previewed market situation.

No	Qualitative Structural Equation	Explanation
1	ADD(HIC, UIC, P)	$HIC = UIC + P_T$. The hedged issuance cost (HIC) is the sum of the unhedged issuance cost (UIC) and the gain from the hedge vehicle (P).
2	ADD(P_T, P, P_t)	$P = P_T - P_t$. The gain from the hedge vehicle is its terminal value (P_T) less the cost paid for it (P_t).
3	ADD(B, P, X) for $B \in (X, \infty)$	$P = max(B - X, 0)$. A major structural equations used to derive the Black-Scholse pricing formula for put options (Smith, 1979). Extracted from node representing put options in ON-ISA network.
4	M$^-(R, B)$	$R \propto^- B$. The relationship between the risk-free interest rate and the value of a bond, which is the put's underlying security. Extracted from node representing bonds in the ON-ISA network.

Table 1: The qualitative structural model for a put option on bonds

4.1 Qualitative Simulation

A qualitative causal analysis that is similar to the one traders use can be done using an algorithm called QSIM (Kuipers, 1986). Given a qualitative structural model of a system and the initial state of parameters in that model, QSIM can be used to describe the qualitative states that system will go through. QSIM propagates the effects of the initial state of parameters through structural connections according to various laws from calculus (i.e., limit analysis) to produce the next qualitative state of each parameter, and of the system as a whole. The qualitative state of a parameter is described by a pair $\langle qdir, qval \rangle$, where $qdir$ is the qualitative direction of change of the value of that parameter (i.e., -1, 0, 1 or decreasing, steady, increasing) over $qval$, which is a qualitative point or region on the real-line. QSIM describes the qualitative behavior of a system by the transitions each of its parameters makes from one qualitative state to another. The transitions of every parameter from one state to another are described by a sequence of pairs $\langle qdir, qval \rangle$ that can be graphically plotted as a two-dimensional piecewise linear function.

To produce the payoff-profile that the purchase or sale of securities in a certain class provides under a given market situation, we can use QSIM with the structural model stored in the node representing that class. Let us use an example to explain how this can be done.

Suppose we are trying to determine if the purchase of a put option on bonds provides a payoff-profile that caps the cost of issuing bonds in case that the risk-free interest rate goes up (see example in Section 2). The input to QSIM includes the set of qualitative equations in Table 1. Equations 1 and 2, in general, model the value of a trader's hedged position. Equation 3 is extracted from the structural model stored in the class of the node representing put option securities in the ON-ISA network. Equation 4 is extracted from the structural model stored in the node representing the underlying bond securities in the ON-ISA network. The input to QSIM also includes the initial state of parameters in these equations, which in this case specifies that the risk-free interest rate, R, is 'increasing' as previewed by the risk manager, and that all other parameters are 'steady'. QSIM's output is a description of the transitions that every parameter makes. The payoff-profile provided by the purchase of put options on bonds is actually described by the transitions of two parameters — the economic variable being hedged (i.e., risk-free interest rate, R) and the value of the trader's hedged position (i.e., hedged issuance cost, HIC).

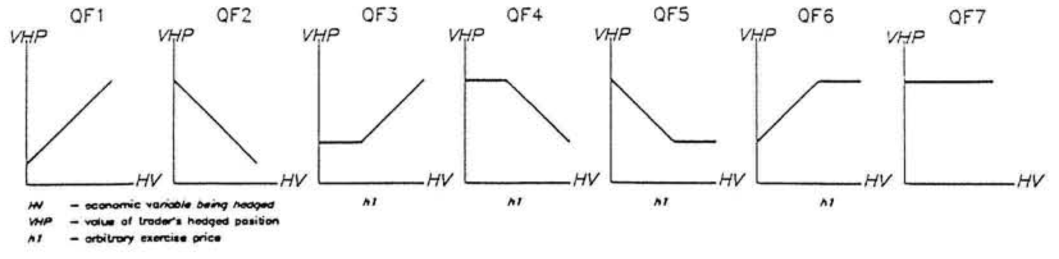

QF1
QF2
QF3
QF4
QF5
QF6
QF7
VHP
HV
AI
HV – economic variable being hedged
VHP – value of trader's hedged position
AI – arbitrary exercise price

Figure 4: Generic payoff-profiles

4.2 Producing Payoff-Profiles

Since QSIM can produce the payoff-profile provided by a hedge vehicle, we have implemented it as a method of class `SECURITY-CLASS`. It is called `Produce-Payoff-Profile`. In the rest of this section we shall refer to this method as QSIM.

To produce the payoff-profile of every generic hedge vehicle, each class node in the ON-ISA network is massaged to do the following: (1) extract the structural equations necessary to run QSIM; (2) run QSIM twice, once for a "buy" action and once for a "sell" action, to produce the payoff-profiles it provides under the previewed market situation; (3) store the two produced payoff-profiles in an attribute called `sell-buy-profiles`; (3) compare the goal payoff-profile against the two stored payoff-profiles; and (5) if a match is not found, mark yourself as a non-feasible hedge vehicle.

Thought a hedge vehicle may provide a different payoff-profile under each different market situation, the payoff-profiles QSIM produces for any type of vehicles is always one of seven payoff-profiles to which we shall refer to as *generic payoff-profiles* (see Figure 4). Remember, however, that it is the use of QSIM that allows us to determine which generic payoff-profile every type of vehicles provides under the market situations being hedged.

Running QSIM for every class of securities in the ON-ISA network can require a lot of computation. We therefore use two heuristics to reduce the number QSIM runs. One heuristic is based on the notion that the sale/purchase of securities whose structural model is not sensitive to the economic variable being hedged provides generic payoff-profile 7, which is meaningless from a hedging stand point. Accordingly, every class node in the ON-ISA network is first massaged to check if the structural model it stores is sensitive to the variable being hedged. If it is not, the node marks itself as a non-feasible hedge vehicle, otherwise it is massaged to run QSIM. The other heuristic is based on the notion that trading is a zero-sum game, which implies that the payoff-profiles for a "buy" and a "sell" action are symmetrical. Therefore, a class node in the network is massaged to run QSIM only for a "buy" action, and then to convert the produced payoff-profile to derive the payoff-profile for a "sell" action.[3]

It is possible that at this point in the configuration phase no generic hedge vehicle provides the goal payoff-profile. In such a case it is necessary to construct compounded hedge vehicles.

[3]It converts a payoff-profile simply by changing the *qdir* in every qualitative state of the parameter representing the value of the hedged position, *VHP*, from 1 (increasing) to −1 (decreasing) and vise versa.

5 Constructing Compounded Hedge Vehicles

The only knowledge stored in the ON-ISA network that can be used to configure compounded vehicles is the payoff-profile of every type of generic vehicles. We can use this knowledge in the same way traders use it to configure compounded payoff-profiles (see example in Section 2). A compounded payoff-profile can be created by permuting a number of generic payoff-profiles. Creating permutations of payoff-profiles that match some goal payoff-profile is a synthesis problem. It is a combinatorial generate-and-test search problem that can be showed to be NP-complete. To solve this problem one must therefore use good heuristics to constrain the generator. To solve this synthesis problem we have developed a technique called *qualitative synthesis.*

5.1 Qualitative Synthesis

Qualitative synthesis uses an algorithm called QSYN (Benaroch & Dhar, 1991c). QSYN considers a payoff-profile to be a two-dimensional qualitative piecewise linear function (hereafter, qfunction). It receives as input a goal qfunction (i.e., goal payoff-profile), G, and a set of generic functions (i.e., generic payoff-profiles), $\mathcal{F}$. It creates one compounded qfunction at a time by algebraically adding two different qfunctions in $\mathcal{F}$. A newly created qfunctions is then compared against G for a match. If it matches part, or all, of G, that qfunction is added to $\mathcal{F}$ with a reference to the two qfunctions in $\mathcal{F}$ that were used to create it. It does the same thing for every pair of different qfunctions in $\mathcal{F}$, including ones containing qfunctions newly added to $\mathcal{F}$. By doing so QSYN finds all possible configurations of compounded payoff-profiles that match the goal payoff-profile.

We define the algebraic sum of two qfunctions as follows. A qfunction is a set of qualitative states of variable *VHP* as a function of variable *VH*. An element of a qfunction is described symbolically as $((HV\ (qdir, qval))(VHP\ (qdir, qval)))$, where $qdir \in \{-1, 0, 1\}$ (i.e., {decreasing, steady, increasing}) and *qval* is a qualitative point or range on the real-line. Assume the existence of two qfunctions QF_i and QF_j, each with n elements, and assume that every pair of corresponding elements in QF_i and QF_j is over the same HV-*qval*. The algebraic sum $QF_i + QF_j$ is a new qfunction QF_{ij} with the same n elements in QF_i, except for the VHP-*qdir* in every element which is the sum of VHP-*qdirs* in every pair of corresponding elements in QF_i and QF_j. For example, the sum of the following pair of corresponding elements k in QF_i and QF_j is

$$\begin{aligned}
QF_i(k) &= \texttt{((HV (1 (h1,h2))) (VHP (1 (v1,v2))))} \\
QF_j(k) &= \texttt{((HV (1 (h1,h2))) (VHP (}-1\texttt{ (v1,v2))))} \\
QF_i(k) + QF_j(k) &= \texttt{((HV (1 (h1,h2))) (VHP (0 (v1,v2))))}.
\end{aligned}$$

Saying that the sum of two qfunction $QF_i + QF_j$ matches part, or all, of a qfunction G means that $QF_i + QF_j$ is equivalent to part, or all, of G. QSYN tests for such equivalence by iteratively checking if $QF_i(k) + QF_j(k)$ is equivalent to $G(k)$ for triplet k of corresponding elements. We say that $QF_i(k) + QF_j(k)$ is equivalent to $G(k)$ for triplet k of elements if: (1) the HV-*qval* in $G(k)$ is contained or equal to the HV-*qvals* in $QF_i(k)$ and in $QF_j(k)$; and (2) the VHP-*qdir* in $G(k)$ is equal to the sum of VHP-*qdirs* in $QF_i(k)$ and $QF_j(k)$.

Suppose we are trying to synthesize the ratio-spread payoff-profile denoted G in Figure 5. It is clear from the example in Section 2 that G can be synthesized from generic payoff-profiles 3 and 4 denoted QF_3 and QF_4 in Figure 5. Yet, the sum $QF_3 + QF_4$ is not equivalent to G. This is because both QF_3 and QF_4 are each representing a whole class

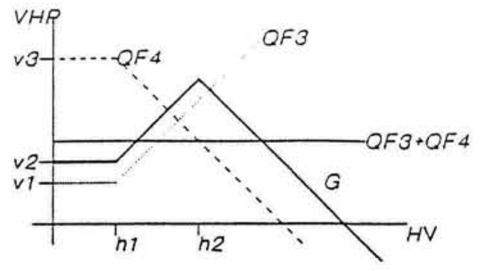


Figure 5: Synthesizing a ratio-spread payoff-profile

of qfunctions with the same general shape. For example, QF_4 represents a whole class of qfunctions whose first element is over an arbitrary range of HV values, $(0, h_i) \in (0, -\infty)$, and whose second element slope is in the range $(0, -\infty)$. What is therefore necessary is to find one qfunction in each class of qfunctions that should be used to synthesize G. Rather than try every qfunction in each of these infinite classes of qfunctions, QSYN can apply heuristic operators to find the specific qfunction needed from each class. We have defined two synthesis operators on qfunctions that help QSYN to narrow down the search space of permutations of qfunctions. These are called *STRETCH* and *STEEPEN*.

5.1.1 Operator *STRETCH*

Suppose QSYN attempts to synthesize qfunction G in Figure 6a. QSYN compares the sum of the second pair of corresponding elements in QF_3 and QF_4 against the second element in G. Clearly, $G(2)$ is not equivalent to $QF_3(2) + QF_4(2)$ because the VHP-*qdir* in $G(2)$ is not equal to the sum of the VHP-*qdirs* in $QF_3(2)$ and $QF_4(2)$. The fact that the VHP-*qdir* in $G(2)$ is equal to the sum of VHP-*qdirs* in $QF_3(2)$ and $QF_4(1)$, however, suggests that a modified version of QF_4, denoted QF_4' in Figure 6b, in which the first element is stretched over the HV-*qval* $(0, h_2)$, is more likely to contribute to the synthesis of G. QSYN has the ability to stretch a qfunction element in such a way using operator *STRETCH*.

Operator *STRETCH* receives two parameters — a qfunction, QF, and an integer, k, designating the number of an element in QF — and works as follows. Assume three qfunctions QF_i, QF_j, and G such that $G(k)$ is equivalent to $QF_i(k) + QF_j(k)$ for triplet k of corresponding elements. Assume also that $G(k+1)$ is not equivalent to $QF_i(k+1) + QF_j(k+1)$ for triplet $k+1$ of elements only because the VHP-*qdir* in $G(k+1)$ is not equal to the sum of VHP-*qdirs* in $QF_i(k+1)$ and $QF_j(k+1)$. If the VHP-*qdir* in $G(k+1)$ is equal to the sum of VHP-*qdirs* in $QF_i(k)$ and $QF_j(k+1)$, operator *STRETCH*$(QF_i; k)$ can be used to stretch element k in QF_i over a HV-*qval* which is the union of the HV-*qvals* in elements k and $k+1$ of G. Operator *STRETCH* can do that by duplicating element k in QF_i, changing its HV-*qval* to be the HV-*qval* in $G(k+1)$, and inserting it after element k in QF_i. For example, *STRETCH*$(QF_4, 1)$ inserts the new element `((HV (1 (h1,h2)))(VHP (0 [v3])))` right after the first element in QF_4 to create QF_4' in Figure 6b.

5.1.2 The *STEEPEN* Operator

Suppose QSYN continues with the synthesis of G in Figure 7a. QSYN compares $G(3)$ against $QF_3(2) + QF_4'(3)$ and concludes that $G(3)$ is not equivalent to $QF_3(2) + QF_4'(3)$ because the VHP-*qdir* in $G(3)$ is not equal to the sum of VHP-*qdirs* in $QF_3(2)$ and $QF_4'(3)$.

However, if QSYN could modify the VHP-*qdir* in $QF_4'(3)$ from -1 to -2 to create a new version of QF_4', denoted QF_4'' in Figure 7b, it can conclude that $G(3)$ is equivalent to $QF_3(2)+QF_4''(3)$. QSYN has the ability to modify the slope of a qfunction element in such a way using operator *STEEPEN*.

Operator *STEEPEN* receives three parameters — a qfunction, QF, an integer, k, designating the number of an element in QF, and an integer, s, designating the slope of a qfunction element — and works as follows. Assume three qfunctions QF_i, QF_j, and G such that $G(k)$ is not equivalent to $QF_i(k)+QF_j(k)$ only because the VHP-*qdir* in $G(k)$ is not equal to the sum of VHP-*qdirs* in $QF_i(k)$ and $QF_j(k)$. If the VHP-*qdir* in $QF_i(k)$ is not zero and if the difference, s, between the VHP-*qdir* in $G(k)$ and the VHP-*qdir* in $QF_j(k)$ is not zero, operator $STEEPEN(QF_i, k, s)$ can be used to increment the VHP-*qdir* of element k in QF_i by s. For example, $STEEPEN(QF_4', 3, -1)$ changes the third element in QF_4' to be `((HV (1 (h2,inf)))(VHP (-2 (minf,v3)))` to create QF_4'' in Figure 7b.

In summary, the use operators *STRETCH* and *STEEPEN* allows QSYN to avoid searching the space of all permutations of generic qfunctions exhaustively. When these operators are used under applicable conditions, they can narrow down the search space significantly. For example, Figure 8 shows the search tree only for permutations of one pair of qfunctions. The emphasized branches in the tree are the ones QSYN chooses to explore. All other branches are readily pruned by operators *STRETCH* and *STEEPEN*.

5.2 Producing Compounded Payoff-Profiles

Since QSYN can use the generic payoff-profiles in Figure 4 to synthesize compounded payoff-profiles that match a goal payoff-profile, we have implemented it in C++ as a method of class `COMPOUNDED-SECURITY-CLASS`. It is called `Synthesize-Payoff-Profile`. In the rest of this section we shall refer to this method as QSYN.

Whenever we start to design a new hedge vehicle, the ON-ISA network has a node called compounded, which represents the class of all compounded securities and that has no instances (see Figure 2). That class node is the only instance of the C++ object class `COMPOUNDED-SECURITY-CLASS`. It can access the method that implements QSYN.

The creation of compounded vehicles that provide the goal payoff-profile is initiated by massaging node compounded in the ON-ISA network to: (1) synthesize the goal payoff-profile from generic payoff-profiles 1 – 6; (2) create a class node called compounded-class$_i$ as an instance of node compounded for every synthesized configuration of the goal payoff-profile, and store that configuration in an attribute called `configuration` in that instance node; and (3) message a newly created instance class node to create its own instances.

A compounded-class$_i$ node, denoted C, proceeds to create its own specific compounded vehicle instances as follows. Node C messages every other class node representing generic securities in the network to indicate its location, if one of the payoff-profiles stored in its attribute `sell-buy-profiles` has the same general shape of one of the generic payoff-profiles stored in attribute `configuration` of node C. Assuming that class nodes A and B have responded to that message, node C connects itself to both nodes using a CONFIGURATION link, and creates for itself one instance node representing a specific compounded hedge vehicle for every combination of instance nodes of A and B representing an instance security (see Figure 9).

Synthesized configurations are actually made from generic payoff-profiles that were pos-

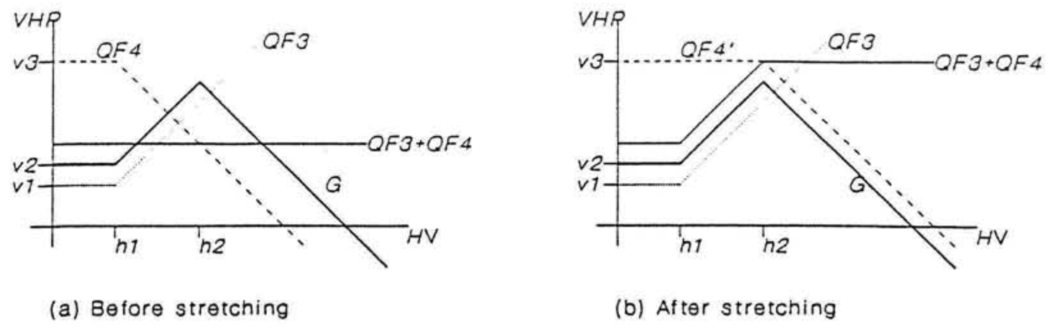


Figure 6: Applying the *STRETCH* operator

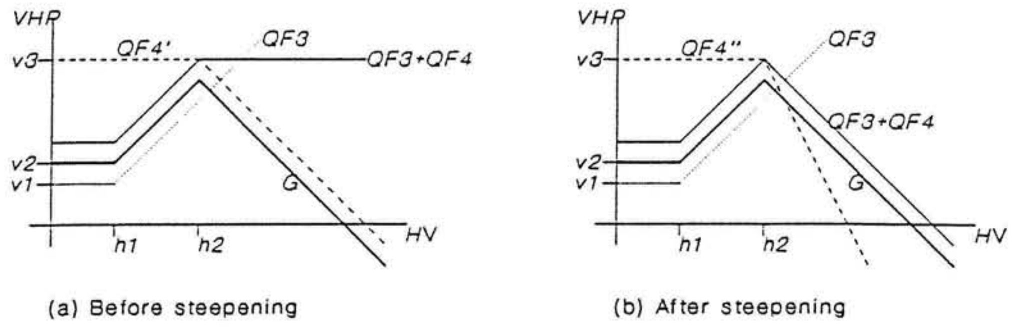


Figure 7: Applying the *STEEPEN* operator

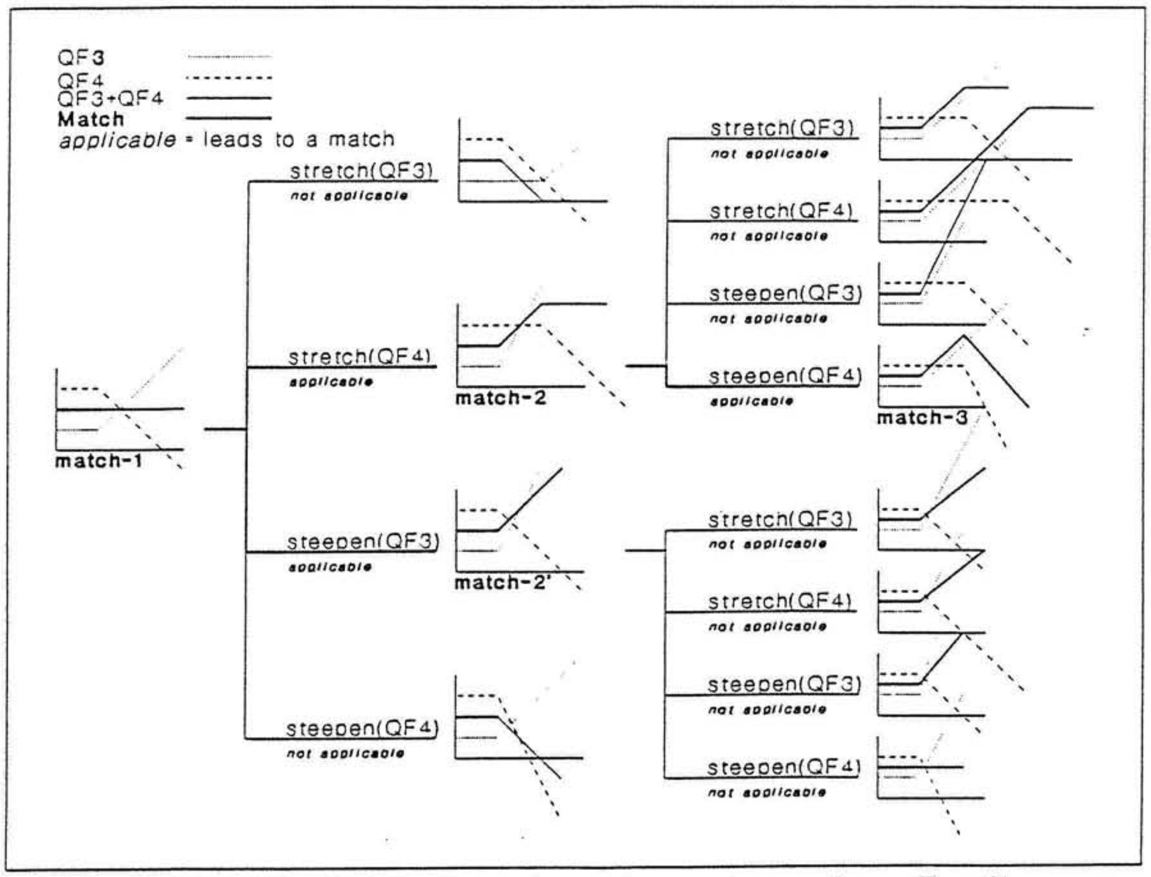


Figure 8: Part of QSYN's search tree for a ratio-spread payoff-profile

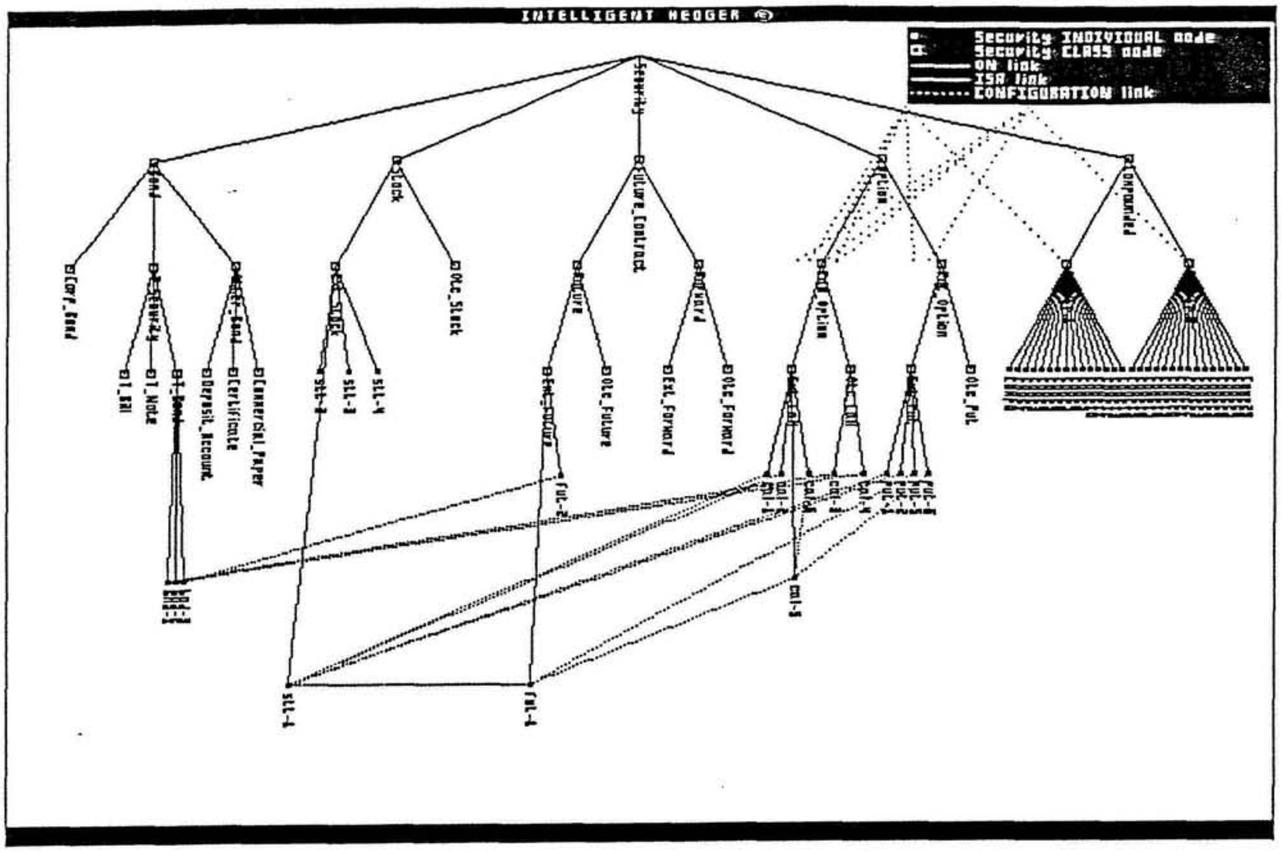


Figure 9: An ON-ISA network with compounded securities

sibly modified by operators *STRETCH* and *STEEPEN*. These modified payoff-profiles provide important information about the nature of a type of compounded hedge vehicles that was created according to a specific configuration. For example, consider the ratio-spread payoff-profile QSYN synthesizes from generic payoff-profiles QF_3 and QF_4'', the modified version of QF_4, which look as follows:

```
QF3   = ( ((HV 1 (0 ,h1 )) (VHP  0 [v1]       ))
          ((HV 1 (h1,inf)) (VHP  1 (v1,inf ))))
QF4'' = ( ((HV 1 (0 ,h1 )) (VHP  0 [v2]       ))
          ((HV 1 (h1,h2 )) (VHP  0 [v2]       ))
          ((HV 1 (h2,inf)) (VHP -2 (minf,v2))))
```

These two have the same general shape of a buy call option on S and a sell option on S, respectively. QF_3 and QF_4'' tell us two other things. Firstly, the exercise price of the purchased call option, h_1, should be smaller than the exercise price of the sold call option, h_2. Secondly, the absolute value of the VHP-*qdir* of QF_4'' (i.e., 2) means that we must sell two or more call option on S to provide the ratio-spread payoff-profile.

6 Discussion

A representation that is designed based on knowledge requirements of the problem domain is more likely to work better than some general purpose representation, such as the rule-based one. Though the design and implementation of a specialized representation involves additional effort in the early part of the development process, it can offer benefits that usually outweigh the additional effort.

While developing INTELLIGENT-HEDGER we have realized that an object-oriented representation such as the one described in this paper offers some extremely valuable features. These features are especially important when the problem domain involves a lot of knowledge that is constantly changing, and requires reasoning with deep knowledge. The major features we have already discussed can be summarized as follows. Firstly, the support of direct reference to basic domain entities and structural relationships between them enables reasoning from first-principles. Secondly, the use of inheritance minimizes the amount of effort needed to set up a KB and to keep it up-to-date. Thirdly, the use of object-oriented concepts provides increased modularity and flexibility in support of an open KB architecture.

There are other features of the object-oriented representation that were not discussed in this paper, but are no less important the ones above. Based on our experience so far, three of these features can be summarized as follows:

- Knowledge modeling and acquisition: modeling deep knowledge using the object-oriented paradigm appears to be natural and intuitive to an expert and to a knowledge engineer. It can therefore reduce the amount of effort needed to elicit knowledge, especially in domains where much of the deep domain knowledge can be elicited from documented sources.
- Transferability of knowledge: the deep knowledge involved in solving a specific problem is often applicable with other related problems in the domain. Deep knowledge that is stored in an object-oriented representations is encapsulated, and therefore more portable. This feature can be extremely critical in determining the ability of a system to share its knowledge with other related systems.
- Ease of control: the development of large knowledge-based systems can often involve complex control issues. In an object-oriented representation the ability of an object to communicate in an almost natural way with other related objects, and the natural hierarchies that are often found between types of objects in a domain can eliminate many of the control issues encountered in other representations.

The object-oriented representation used in INTELLIGENT-HEDGER has one major limitation. Most domains involve, to one extent or another, expert knowledge in the form of situation-action rules that can not be derived by reasoning from first-principles. Integrating such surface knowledge into our representation can be complex. We are currently exploring a number of solutions to this problem. One that seems to be the most promising for domains such as trading, where we often find experts that specialize only in one type of securities, can be summarized as follows. It may be feasible to encode surface knowledge about every type of securities in a 'small' knowledge base that is stored as an attribute of the object representing that type of securities, and to implement a specialized inference engine with that type of surface knowledge as a method of the object. Accordingly, whenever an object cannot derive the response to another object from deep domain knowledge about itself, that object can invoke its specialized inference engine to try and derive a respond using the surface knowledge it stores.

References

[1] Baecher, E., and Goodman, S.L., *The Goldman Sachs Guide to Hedging Corporate Debt Issuance*, Financial Strategies Group, Goldman Sachs, 1988.

[2] Benaroch, Michel, and Dhar, Vasant, 1991a, "An Intelligent Assistance for Financial Hedging," *Proceedings of the 7th IEEE Conference on AI Applications.*

[3] Benaroch, Michel, and Dhar, Vasant, 1991b, "A Knowledge-Based Approach to Solving Hedge Design Problems," *IEEE Proceedings of the First International Conference on AI Applications on Wall Street*, Forthcoming.

[4] Benaroch, Michel, and Dhar, Vasant, 1991c, "Qualitative Synthesis of Configurations of Two-Terminal Systems Based on Desired Behavior," CRIS Working Papers Series, New York University.

[5] Elton, J.E. and Gruber, J.M., *Modern Portfolio Theory and Investment Analysis (3rd edition)*, John Wiley & Sons, Inc., 1987.

[6] Kuipers, B., "Qualitative Simulation," *Artificial Intelligence*, 29:289–338, 1986.

[7] Mittal, Sanjay, and Araya, Agustin, "A Knowledge-Based Framework for Design," *Proceedings of the 5th International Conference on AI, AAAI-86*, Philadelphia, 1986.

[8] Rambaugh, J., Blaha, M., Premerlani, W., Eddy, F., and Lorensen, W., *Object-Oriented Modeling and Design*, Prentice Hall, Englewood Cliffs, NJ, 1991.

[9] Smith, W.C., "Applications of Option Pricing Analysis," In James L. Bicksler, eds., *Handbook of Financial Economics*, North-Holland, New York, 1979.

[10] Sutton, William, *Trading in Currency Options*, New York Institute of Finance, 1988.

[11] Williams, Brian, C., "Interaction-Based Invention: Designing Novel Devices from First Principles," *AAAI Proceedings*, 1990.