Making Risk Management Systems Smart

Dr Jean-Noël DORDAIN, jnd@clipper.ens.fr

Niladri SINGH, singh@clipper.ens.fr

Cahier du CEREG 200002

Abstract

This paper introduces concepts used to rigorously model financial transaction processes within a front to back office integrated risk management system. This unified modelling offers a framework particularly well adapted to the optimisation of the computation of financial product quantitative parameters. Furthermore, it allows to keep track of the various computations performed.

Abstract

Cet article développe de nouveaux concepts qui permettent de modéliser de façon rigoureuse l'intégration front / back office des transactions financières au sein d'un logiciel de gestion du risque. Cette modélisation offre un cadre particulièrement bien adapté l'optimisation et à la traçabilité des calculs de paramètres quantitatifs définissant les produits financiers.

Introduction

This paper deals with the efficient design of front to back office integrated risk management systems. Having practised a number of risk management systems, the authors have experienced the necessity of furnishing the financial processes constitutive of the trading and risk management tasks with a detailed structure amenable to a rigorous modelling and implementation treatment.

Data descriptions of financial products are complex and, within a given portfolio, the data describing two financial products may be interlaced in a very intricate way. Furthermore, financial portfolios usually contain very large amounts of data by common standards. Hence, to address the issues encountered in quantitative finance, one needs much more than the simple combination of a database manager and of a pricing functions library interfaced through their respective APIs. Therefore, a risk management system cannot be reduced to its financial products pricing and data management functions.

One must necessarily consider abstract objects when assessing quantitative parameters of financial objects. By abstract objects, we mean intermediate computational data that are neither inputs nor outputs of a risk management system. An example is given by the Hull & White interest rates framework, where the volatility input is the implied Black & Scholes volatility term structure computed from cap & floor market prices but the true computational volatility is the gaussian equivalent term structure. Another example of the pervasive nature of abstract data in financial computations is given by the need, when performing a very large number of operations, to extract patterns from the portfolio that make optimisation possible with respect to a given set of criteria.

The aim of our approach is to furnish the pricing routines scheme and the financial products management scheme with a unified algebraic structure. This additional level of structuring makes it possible to integrate a new applicative entity dedicated to operations on the algebraic layer. This new applicative entity permits both total automatisation and fully optimised management of the financial processes from front to back office.

The present paper is structured as follows.

• In Part 1, we give a detailed analysis of the usual pitfalls of risk management systems. This analysis leads us to formulate basic requirements that a

risk management system must meet.

• In Part 2, we give an exhaustive description of the algebraic structure above mentioned. The building blocks of this structure are: the financial types system, the attributes framework, the relation framework and the pricing models. We then show how these elements can be combined to yield product graphs: algebraic representations of real financial portfolios.

• In Part 3, we show how this algebraic framework can be used to build algorithms performing efficient financial computations. Among the important features of our algorithms we shall specifically concentrate on the following: avoiding useless computations, avoiding redundant computations and performing computations optimised with respect to a given set of criteria.

• In the conclusion, we turn our attention to parallel computing and we show that our algorithms permit straightforward parallelisation of the computation processes.

1 Risk management systems

1.1 Common pitfalls of risk management systems

The two most common pitfalls of risk management system implementations appear to be:

• the system is an heterogeneous collection of independent software – where each application is focused on a single area of expertise within the bank business. The major drawback of this approach is that when two applications with different views of the business need to communicate, the interface layer that has to be built must incorporate not only a translation module but also a computation module.
Consider for instance a front-office dedicated pricing system and a risk controlling VaR oriented system. The historical volatility used for front office pricing is expressed on a daily to monthly basis whereas the volatility used for risk controlling is expressed on a monthly to annual basis. Thus, a front-office / risk-control interface layer is a full fledged application able to compute volatilities on various time bases. As shown in **figure 1** this interface layer is actually the juxtaposition of an application and two interface layers.

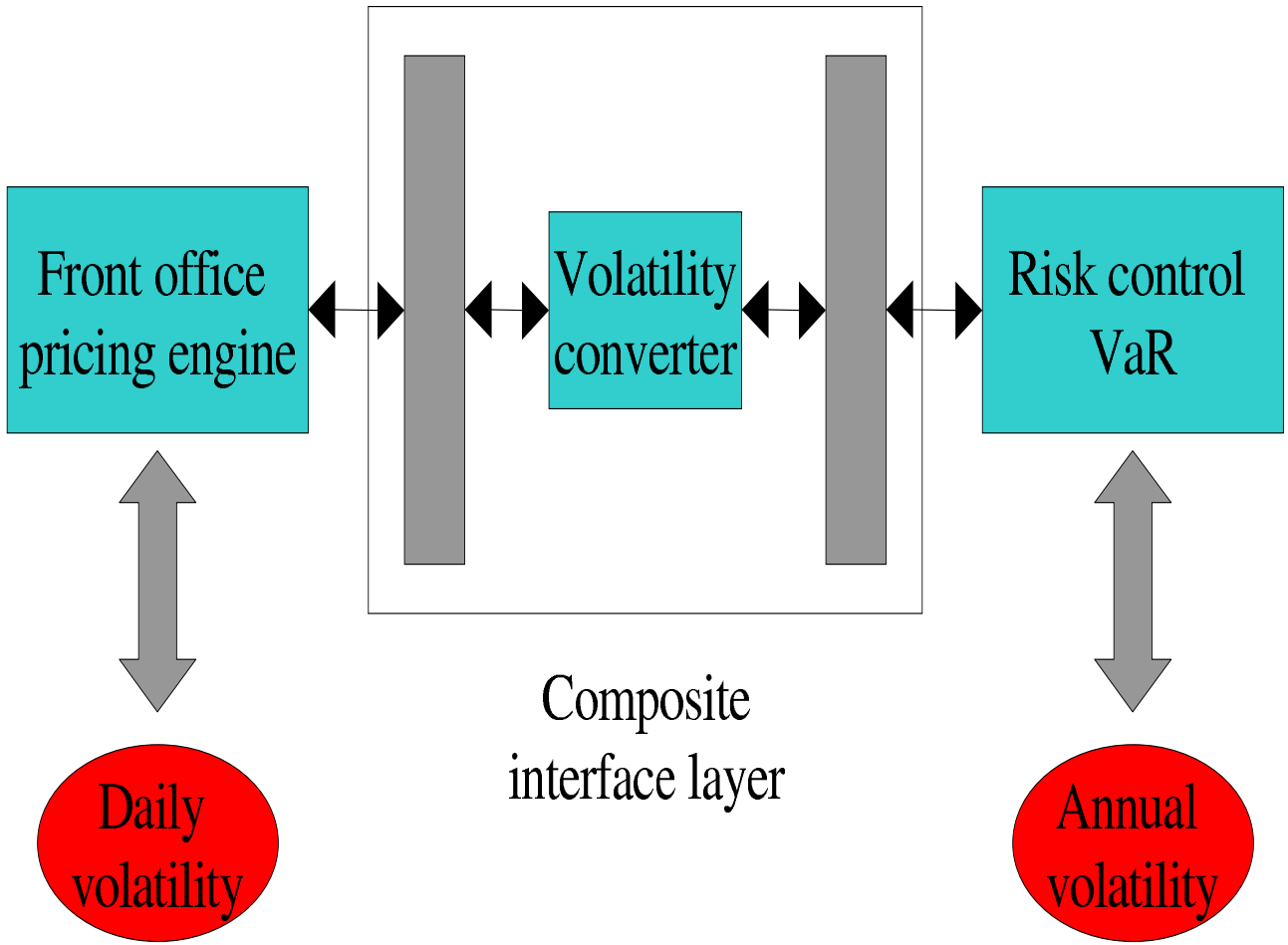

Front office
pricing engine
Volatility
converter
Risk control
VaR
Composite
interface layer
Daily
volatility
Annual
volatility

Figure 1

• One of the risk management applications is much larger than the others and the design of this central application commands and limits that of all the other applications. Usually, such a situation arises when a bank out-sources its front office system and buys an integrated risk management solution. For historical reasons, many off-the-shelf risk management systems deal exclusively either with basic financial products – plain vanilla and other simple options that are standardised and massively traded – or structured OTC financial products – with low traded volumes and sophisticated pricing methods. Off-the-shelf systems that have been designed to answer needs arising from the trading of basic financial products are data focused whereas systems that have been designed to answer needs arising from the trading of structured financial products are computation focused. Hence, either data are well processed but pricing methods are rudimentary or pricing methods are well suited to all kinds of products traded but the data treatment is primitive.

1.2 Risk management requirements

A truly integrated and adaptable system must satisfy the following requirements:
• all the bank's know-how is optimally used, each skill being integrated with respect to its particular field of competence,
• the system is complete, the same data are coherently used throughout the bank – feed, update, computation and historization –,
• the system is clearly subdivided into mission focused elementary blocks,
• whenever a functionality must be added, the various blocks can be upgraded easily and independently, the development process that is required can be clearly identified and its price can be predicted to a satisfactory level of accuracy.

At a department level, banks are usually structured as follows:

• the front-office,
• the research – financial analysts and quantitative analysts –,
• the middle-office,
• the back-office.

The area of expertise of the various departments are:

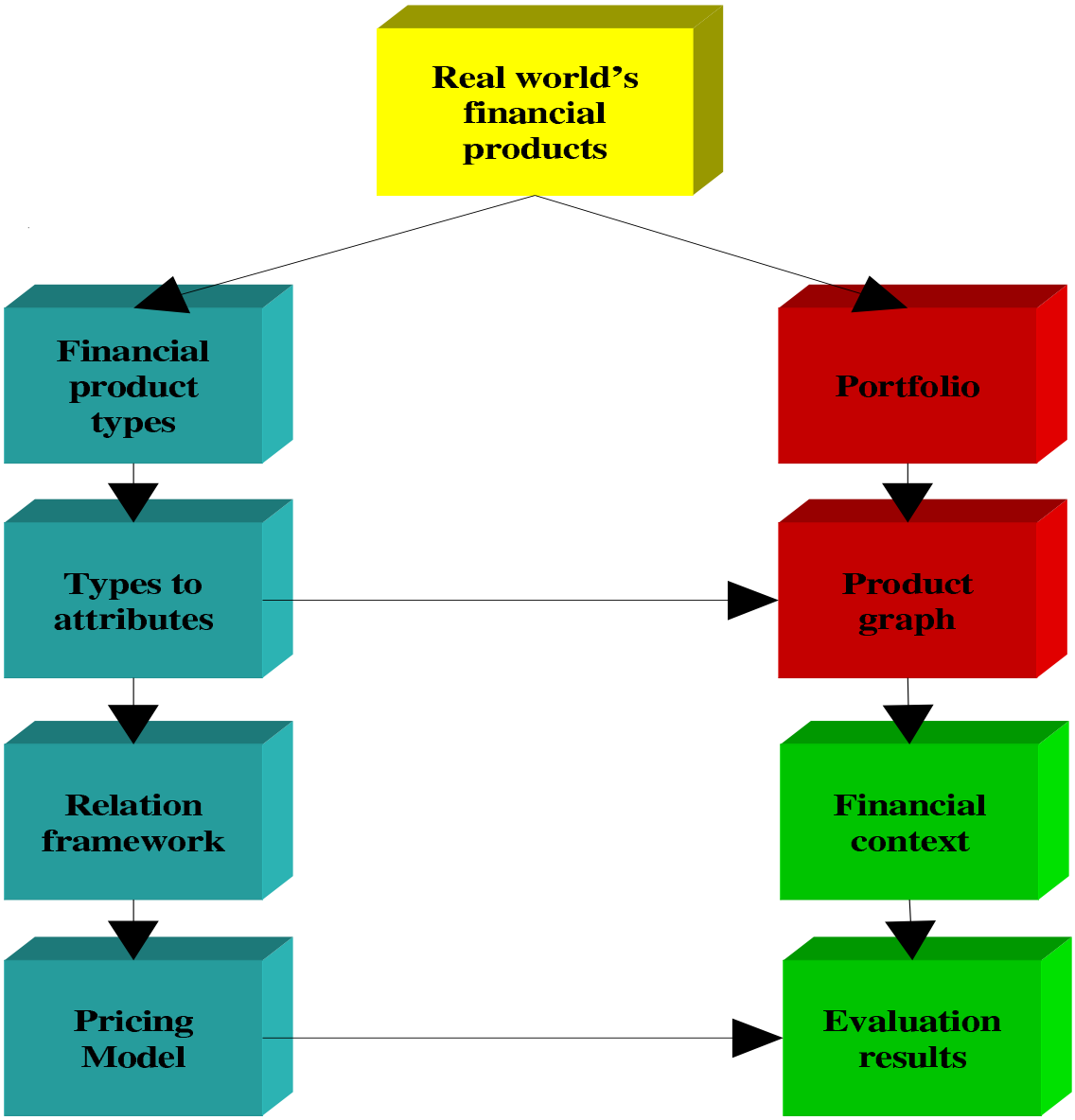
• to build, to structure and to manage a portfolio is the front-office business,

• for a given financial product, to define the features and the risk factors relevant to pricing and general risk management is the business of financial analysts within the research department,
• to design and to implement the various evaluation algorithms is the role of quantitative analysts,
• to manage and to control the data feed is one of the middle-office tasks,
• to precisely define the standard features of global financial products is the back-office role.
A global risk management system must incorporate all the value that is created by the various departments of the bank and provide a global representation of the bank's traded assets in order to permit the evaluation of their prices and of their risk parameters, local and global.
Hence, the architecture of a global risk management system can be unbundled into elementary blocks – competence-wise and function-wise – as represented in **figure 2**.

1.3 Risk management abstraction blocks

In **figure 2**, we have represented the dependencies – temporal and structural – between the various blocks of a risk management system that would meet the requirements that we have already formulated. Namely:
• the **Financial product types** block parallels the standardisation of financial products operated by the back-office,
• the **Types to attributes** block parallels the definition of financial products features and risk factors operated by the financial analysts within the research department,
• the **relation framework** block represents the relations existing between the various financial variables, irrespective of the particular pricing algorithms that one chooses to associate with these relations, whereas the **pricing model** block is the set of financial pricing algorithms that can be used for valuation purposes. Thus these two blocks parallel the task achieved by quantitative analysts within the research department,
• the **portfolio** block represents the portfolio managed by the front-office,
• the **financial context** block represents the data that have been validated by the middle-office,

Figure 2


Real world's financial products
Financial product types
Types to attributes
Relation framework
Pricing Model
Portfolio
Product graph
Financial context
Evaluation results

• the **product graph** block is the processed image of the bank's traded assets within the risk management system, whereas the **Results** block contains the prices and the risk parameters that are used to conduct the bank business.

2 The algebraic set-up

We shall now proceed with a formal algebraic description of the issues raised by the design of a risk management system.

2.1 Financial product types

A **system of financial product types** is a triple $(\mathcal{G}, \mathcal{N}, f)$ where:

• The set $\mathcal{N}$ is the **set of names**,
• The set $\mathcal{G}$ is a set of graphs called the **set of types**,
• The map $f : \mathcal{G} \to \mathcal{N}$ is an into map called the **naming function**.

Furthermore, $\mathcal{G}$ possesses the following properties:

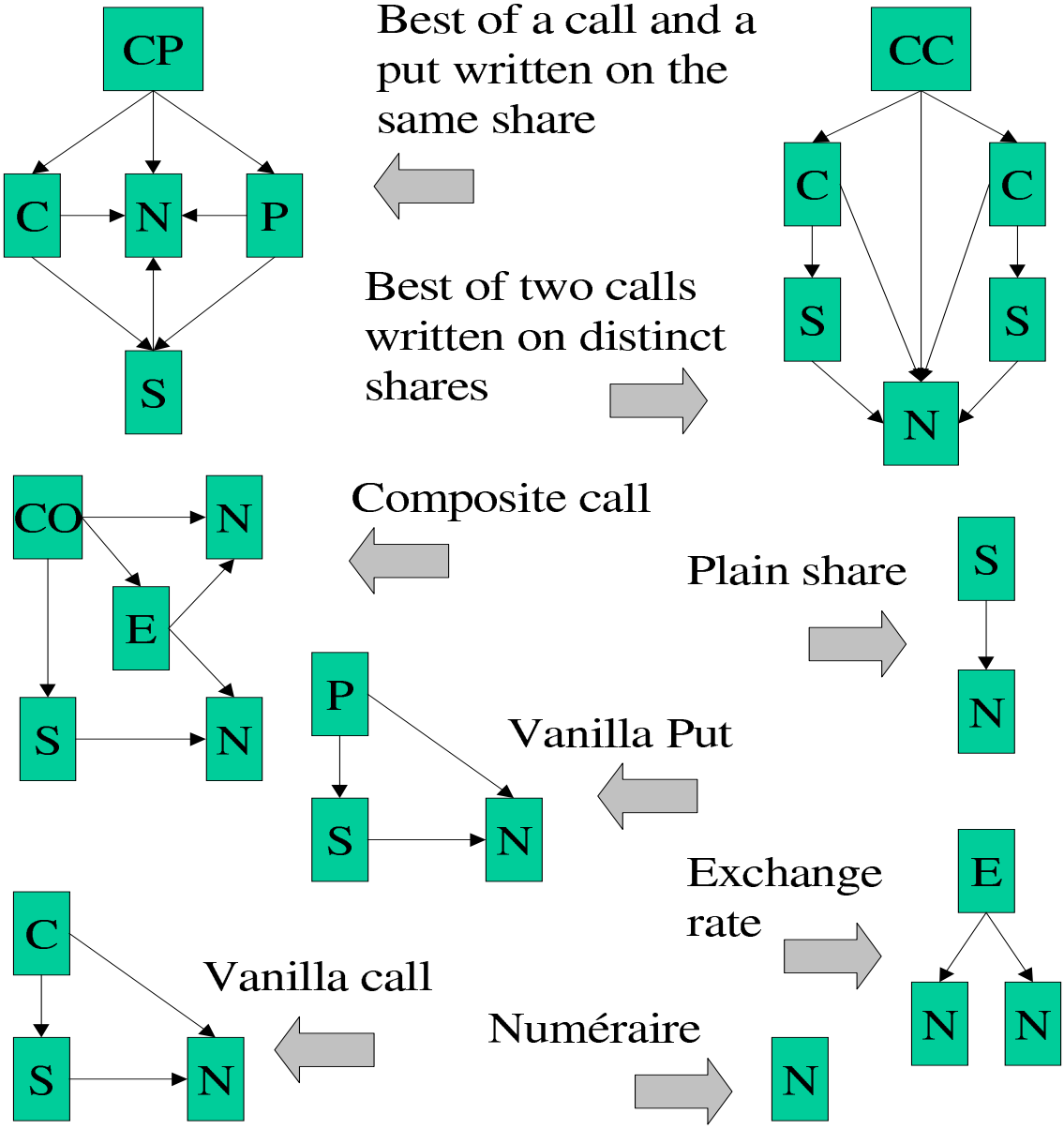
• the elements of $\mathcal{G}$ are directed, acyclic and rooted graphs,
• the set $\mathcal{G}$ is closed. By closed, we mean that given $g \in \mathcal{G}$ and a node n of g, the sub-graph $g(n)$ of g starting at n – i.e. the subset of g of all elements lower than n – is also an element of $\mathcal{G}$.

To deal with multiple currency products, we impose the following conditions on the triple $(\mathcal{G}, \mathcal{N}, f)$:

• The set $\mathcal{N}$ possesses two elements named "Numeraire" and "Exchange Rate",
• every type $g \in \mathcal{G}$ such that $f(g) =$ "Numeraire" is minimal – i.e. it possesses a unique node –,
• every non minimal $g \in G$ with $f(g) \neq$ "Exchange Rate" possesses a unique edge pointing towards a minimal node,
• every $g \in \mathcal{G}$ with $f(g) =$ "Exchange Rate" possesses exactly two edges pointing towards two distinct minimal nodes.

An example of a closed system of financial product types with multiple currencies is given in **figure 3**. The order relation on a graph of the product types set corresponds to the relation on real financial assets defined by "A is an underlying of B". Thus, at the type level, there exists an edge from the type of B pointing towards the type of A.

Figure 3


CP
C
N
P
S
Best of a call and a put written on the same share
CC
C
C
S
S
N
Best of two calls written on distinct shares
CO
N
E
S
N
Composite call
Plain share
S
N
P
S
N
Vanilla Put
Exchange rate
E
N
N
C
S
N
Vanilla call
Numéraire
N

In **figure 3**, the set of values assumed by the naming function is

$$\{N, E, S, C, P, CO, CP, CC\}$$

where the symbols have the following financial interpretations:
– the symbol N stands for the type of a "Numeraire",
– the symbol E stands for the type of an "Echange rate",
– the symbol S stands for the type of a "Share",
– the symbol C stands for the type of a "Call",
– the symbol P stands for the type of a "Put",
– the symbol CO stands for the type of a "Composite call",
– the symbol CP stands for the type of a "Best of call-call",
– the symbol CC stands for the type of a "Best of call-put".

2.2 Attribute framework

An attribute framework is a triple $(\mathcal{N}, \mathcal{A}, Att)$ where:
• the set $\mathcal{N}$ is a set called the **set of the names**,
• the set $\mathcal{A}$ is a set called the **set of all attributes**,
• the set $\mathcal{P}(\mathcal{A})$ is the power-set of $\mathcal{A}$ and the map

$$Att : \mathcal{N} \to \mathcal{P}(\mathcal{A})$$

is the **attribution map**.

Given a system of financial product types $(\mathcal{G}, \mathcal{N}, f)$ and an attribute framework $(\mathcal{N}, \mathcal{A}, Att)$, with the same set of names, we can build the triple

$$\mathcal{PF} = (\mathcal{G}, \mathcal{P}(\mathcal{A}), Att \circ f)$$

Whence the map $Att \circ f$ can be treated as a new naming function. Such a triple $\mathcal{PF}$ is called a **pricing framework**.

The **set of simple attributes** of a given product type g in the pricing framework $(\mathcal{G}, \mathcal{P}(\mathcal{A}), Att \circ f)$ is the set

$$\mathcal{S}_g = \{(x, y) \in \mathcal{G} \times \mathcal{A} |\ x \in g \text{ and } y \in Att \circ f(x)\}$$

where $x \in g$ means that x is a node of g.

2.3 Relation framework

Given a poset $(P, \leq)$ a **closure on** P is a map $Q : P \to P$ for which the following properties hold:

- the map Q is idempotent – i.e. $Q \circ Q = Q$ –,
- given any $x \in P$, $x \leq Q(x)$,
- given any $x, y \in P$, $x \leq y \implies Q(x) \leq Q(y)$.

A **simple relation framework** with respect to a product type g is a closure on the power-set $\mathcal{P}(\mathcal{S}_g)$ of the set of simple attributes of g.

Given a product type g and a simple relation framework Q_g for g, the image $Q_g(x)$ of a subset x of the set of attributes $\mathcal{S}_g$ of g represents the set of all attributes that can be computed from x. Thus the three conditions stated above have the following financial interpretation:

- the map Q_g does not create any additional information,
- saying that x can be computed from x is a tautology,
- if y contains more information than x, then $Q_g(y)$ contains more information than $Q_g(x)$.

Given a product type g and a simple relation framework Q_g for g, an attribute $a \in \mathcal{S}_g$ of g is said to be **well-defined** with respect to $x \subset \mathcal{S}_g$ if there exists $z \subset x$

$$\forall y \subset \mathcal{S}_g,\ a \in Q_g(y) \implies y \subset Q_g(z)$$

This condition ensures that if there are two different ways to compute a from x, both computations must, at the relational level, lead to the same result.

Given a pricing framework $(\mathcal{G}, \mathcal{P}(\mathcal{A}), Att \circ f)$, a **relation framework** is a family $\mathcal{RF}$ of simple relation frameworks

$$\mathcal{RF} = (Q_g)_{g \in \mathcal{G}}$$

where for all $g \in \mathcal{G}$, Q_g is a simple relation framework for g.

2.4 Pricing model

Given a relation framework $\mathcal{RF}$, a **pricing model** $\mathcal{PM}$ is a triple

$$\mathcal{PM} = \left((Dom_a)_{a \in A}, (F_x^g)_{g \in \mathcal{G}, x \subset \mathcal{S}_g}, \perp\right)$$

such that:

• all the elements of the family $(Dom_a)_{a\in A}$ are sets,
• for all $g \in \mathcal{G}$, $x \subset \mathcal{S}_g$, F_x^g is a map

$$F_x^g : Dom(x) \longrightarrow Dom(Q_g(x)) \cup \{\perp\}$$

where for all $x \subset \mathcal{S}_g$ the set $Dom(x)$ is defined by

$$Dom(x) = \prod_{u\in x} Dom_{pr_2(u)}$$

• for all $g \in \mathcal{G}$, $x \subset \mathcal{S}_g$, $\perp \notin Dom(x)$.

Given a type $g \in \mathcal{G}$, the set $Dom(x)$ represents the set of all values that can be **syntactically** assumed by attributes of x. Given a certain assignation $a \in Dom(x)$, the function F_x^g checks whether the assignation is **value-wise coherent** and maps a to:

– either an element of $Dom(Q_g(x))$ if a is a value-wise coherent assignation,
– or the symbol $\perp$ if a is not a value-wise coherent assignation.

Thus, the function F_x^g performs two operations: it checks whether the data passed to it is coherent and computes the maximal set of data that can be determined from x.

For the computation results to be coherent we must further impose a **transitivity condition** on the set of pricing functions (F_x^g) (see **figure 4**). Given a type $g \in \mathcal{G}$ and two subsets x, y of attributes of g with $x \subset y$ denote $\Pi^g_{x\to y}$ the natural projection – component-wise – from $Dom(x)$ to $Dom(y)$ – i.e.

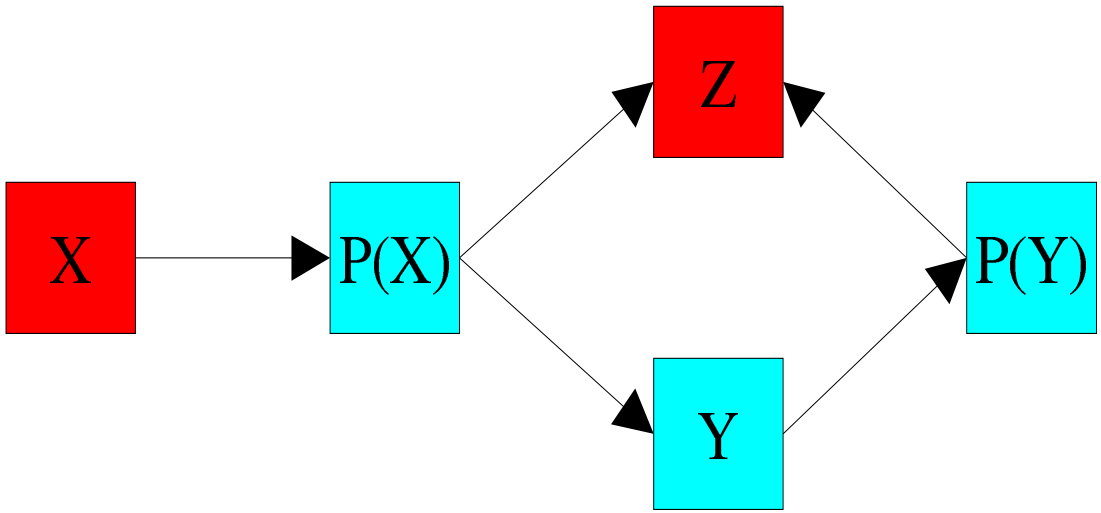
$$\Pi^g_{x\to y} : Dom(x) \longrightarrow Dom(y)$$

Given a type $g \in \mathcal{G}$ and three subsets x, y and z with $y \subset P(x)$ and $z \subset P(y)$, we say that the transitivity condition holds if

$$\Pi^g_{Q_g(x)\to z} \circ F_x^g = \Pi^g_{Q_g(y)\to z} \circ F_y^g \circ \Pi^g_{Q_g(x)\to y} \circ F_x^g$$

2.5 Financial products

We have already outlined that financial products possess a hierarchical structure. Thus, denoting by $\mathcal{W}$ the **set of all financial products**, there is a natural order relation on $\mathcal{W}$ defined by $a \leq b$ if a is an underlying of b.



Two paths from X to Z :

1) X→P(X)→Y→P(Y)→Z

2) X→P(X)→Z

The two paths lead to the same results

Our hypothesis is that all financial products have a type and that there exists a strictly increasing onto map:

$$\Phi : \mathcal{W} \longrightarrow \mathcal{G}$$

Furthermore the type system is **faithful**, for all financial product $p \in \mathcal{W}$ the sets $\Phi(p)$ and $\{q \in \mathcal{W} | q \leq p\}$ are isomorphic as directed graphs.

2.6 Construction of the product graph

Let $\mathcal{E}$ be the set

$$\mathcal{E} = \{(p, g) \in \mathcal{W} \times \mathcal{G} | g \in \Phi(p)\}$$

and let $\mathcal{R}$ be the relation defined on $\mathcal{E}$ by

$$(p, g)\mathcal{R}(p', g') \Leftrightarrow p \leq p' \text{ and } g = g'$$

Finally let $\mathcal{S}$ be the smallest equivalence relation containing $\mathcal{R}$. We denote π the natural projection

$$\pi : \mathcal{E} \longrightarrow \mathcal{E}/\mathcal{S}$$

A **portfolio** $Port$ is a linear combination of financial products. The base $B(Port)$ of the portfolio is the subset of $\mathcal{W}$ of all financial products appearing in $Port$ with a non-zero coefficient. The **product graph** of the portfolio $Port$ is the image of $B(Port)$ by the map π.
All financial products mapping to the same $x \in \mathcal{E}/\mathcal{S}$ have the same type, therefore we can define a map

$$type : \mathcal{E}/\mathcal{S} \longrightarrow G$$

Thus we can define the set of all attributes of the portfolio $Port$ by

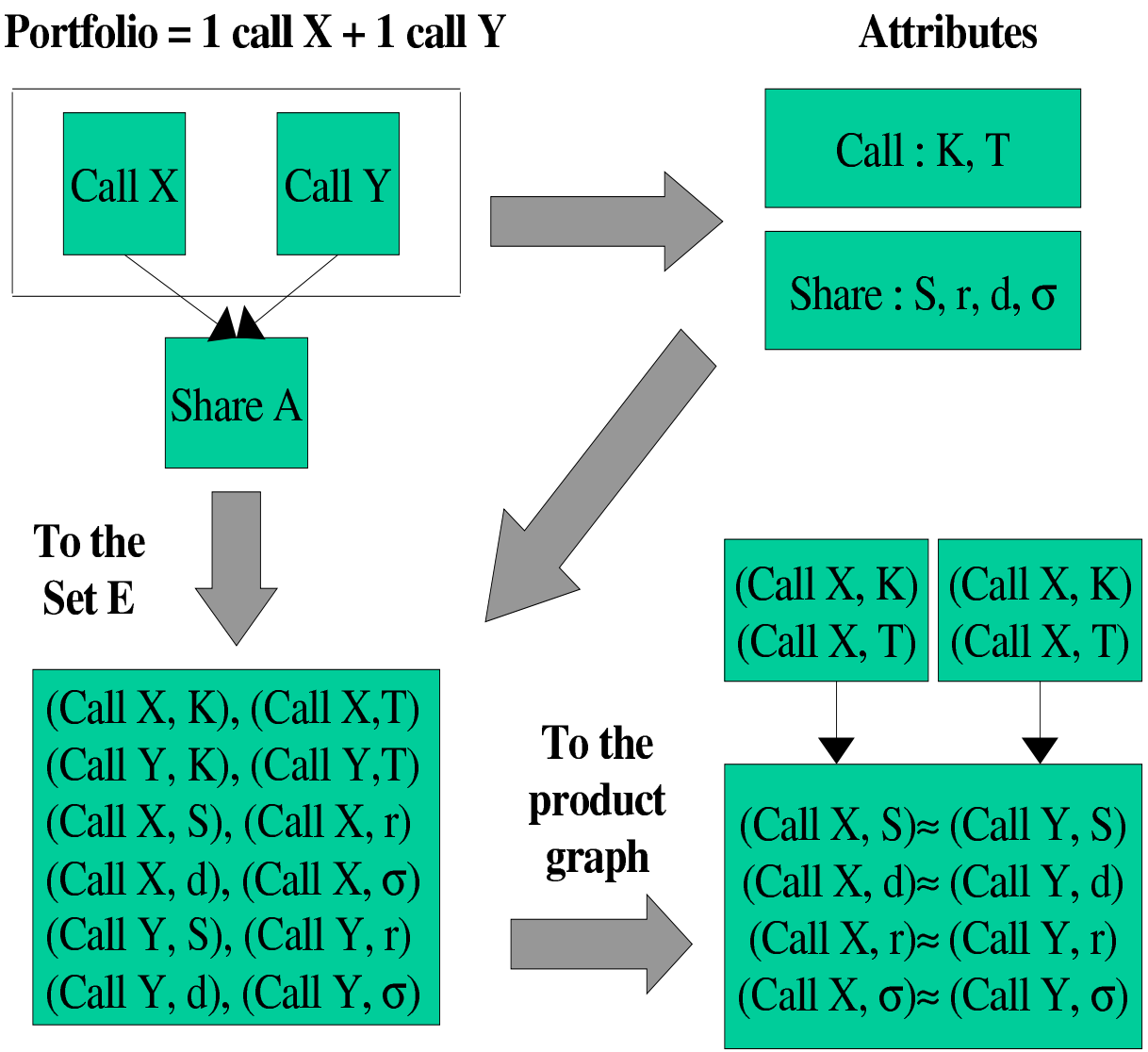
$$\mathcal{S}_{Port} = \{(x, a) \in \mathcal{E}/\mathcal{S} \times \mathcal{A} | x \in \pi(B(Port)), a \in Att \circ f(type(x))\}$$

A example of a product graph construction is given in **figure 5**.

2.7 Financial context

A financial context for the portfolio $Port$ is an element of the set

$$Dom(Port) = \prod_{b \in B(port)} (Dom(b) \cup \{\perp\})$$


Portfolio = 1 call X + 1 call Y
Attributes
Call X
Call Y
Share A
Call : K, T
Share : S, r, d, σ
To the
Set E
(Call X, K), (Call X,T)
(Call Y, K), (Call Y,T)
(Call X, S), (Call X, r)
(Call X, d), (Call X, σ)
(Call Y, S), (Call Y, r)
(Call Y, d), (Call Y, σ)
To the
product
graph
(Call X, K)
(Call X, T)
(Call X, K)
(Call X, T)
(Call X, S)≈ (Call Y, S)
(Call X, d)≈ (Call Y, d)
(Call X, r)≈ (Call Y, r)
(Call X, σ)≈ (Call Y, σ)

The base of the financial context $c \in Dom(Port)$ is the subset of $\mathcal{S}_{Port}$ defined by

$$Base(Port) = \{(x, a) \in B(Port) \times \mathcal{A} | c(x, a) \neq \perp\}$$

3 Computations on the product graph

We have shown how the strong typing we have imposed on financial products has allowed us to avoid financial data layering by providing a unified treatment of computational requests – e.g. the same computational request may contain both the computation of a price and the computation of an implied volatility. One of the key features of our framework is that it does not suffer the usual drawback of flexible systems that are often unstable.
We shall demonstrate in this section that our model is both flexible and reliable. To this purpose we shall focus our attention on algorithms allowing the determination of data by working on the relational framework. Such algorithms express intrinsic properties of the product graph with respect to a given relational framework.
Remember that a computational request is a set of assigned data together with a set of unknown data.
We shall concentrate our attention on threefold algorithms obeying to the following pattern:

• determine what data can be computed from the set of all initial assigned data,

• identify a best computational sequence for every computable data,

• perform the set of selected computations and, after every elementary computation, check the sanity of the intermediate results.

3.1 Reachable data analysis

An attribute is reachable from a financial context if it is in relation, via the relation framework, with the base of that context.
Obviously, given a financial context c, taking the union of its projections on all product types does not yield the set of all reachable data as it does not account for inter-product redundancies within the portfolio.

Furthermore, it is clear that computing all the productions of the complete product graph would immediately yield the set of reachable data for any financial context. However, this approach is obviously untractable as for a product graph with n distinct attributes, it requires the knowledge of all the values assumed by a function defined on the power-set 2^n.
To make our point, let us consider the case of a portfolio consisting exactly of one share, one put and one call. Let S_0, d, σ be the spot, the continuous dividend rate and the volatility of the share, let P_0, K_P and T_P be the price, the strike and the maturity of the put and let C_0, K_C and T_C be the price, the strike and the maturity of the call. We assume that the interest rate r is constant and we adopt the Black & Scholes methodology.
Let c be a context of witch the base is the set of data r, d, S_0, K_C, K_P, T_C, T_P, P_0. On the one hand, adopting the restrictive product-by-product projection approach would only enable us to compute the share volatility – whereas by transitivity, the price of the call also belongs to the set of reachable data. On the other hand, the product graph we have considered, although very small, already possesses 10 distinct attributes so that computing the production of the complete product graph requires a knowledge of $2^{10} = 1024$ associations between sets of attributes.
A computationally efficient solution to the reachable data problem is given by the linear complexity algorithm we shall now describe. This algorithm is an iteration on a three steps procedure: dispatch, expand and update.
• In the dispatching phase, we take the projection of the financial context base on every product in the portfolio. Mathematically dispatching comes to taking for every product in the portfolio the intersection of its set of attributes with the context base. We know that this dispatching process is well defined because the strong typing requirement ensures that two distinct attributes of a given type cannot be mapped to the same attribute of the product graph.
• In the expanding phase, one computes for every product in the portfolio the production of the subset obtained in the dispatching phase by projecting the context base into the product set of attributes. Thus, we exploit product-by-product the information present in the relation framework.
• In the updating phase we take the union in the product graph of the projections of the new product-wise sets of attributes obtained during the "expanding" phase.
The iteration terminates if the result of the updating phase coincides with

the argument of the expanding phase. This means that the context d that has been obtained from the initial context is a fixed point for the iteration procedure

$$x \mapsto \text{ dispatching } \rightarrow \text{ expanding } \rightarrow \text{ updating } \mapsto x'$$

i.e. $d = d'$.
This algorithm (its principle is illustrated in **figure 6**) is meant to determine all the data that can be computed from a given input. However, one usually restricts one's attention to a given subset x of the set of all data. Thus, if the base of a context contains x no supplementary interesting information can be found at the search scope through the dispatch, expand and update procedure. Thus, given an analysis scope x, the termination condition of the dispatch, expand and update iterative procedure may now be written:

$$x \subset d' \text{ or } d = d'$$

Obviously, if no restriction is placed on the analysis scope – i.e. x is the set of all attributes – this termination condition is the same as the first termination condition stated. An example of the algorithm (with no restriction on the scope) is given in **figure 7**.

3.2 Choosing a computation path

Let us consider the update step of the algorithm described above. Observe that if the fixed point context solution of the algorithm has not been reached, then there are in the base of the output context data that are not in the base of the input context. In the dispatch, expand and update procedure described above the interesting point is not how new data items can be computed but what new data items can be computed. Indeed at a given time, the same data may be obtained via the projections of the input context on two distinct financial products. Thus to be able to perform a computation round, we must be able to choose at every time step, and for every reachable data, the product from which it should be computed.

Determination of the computation path

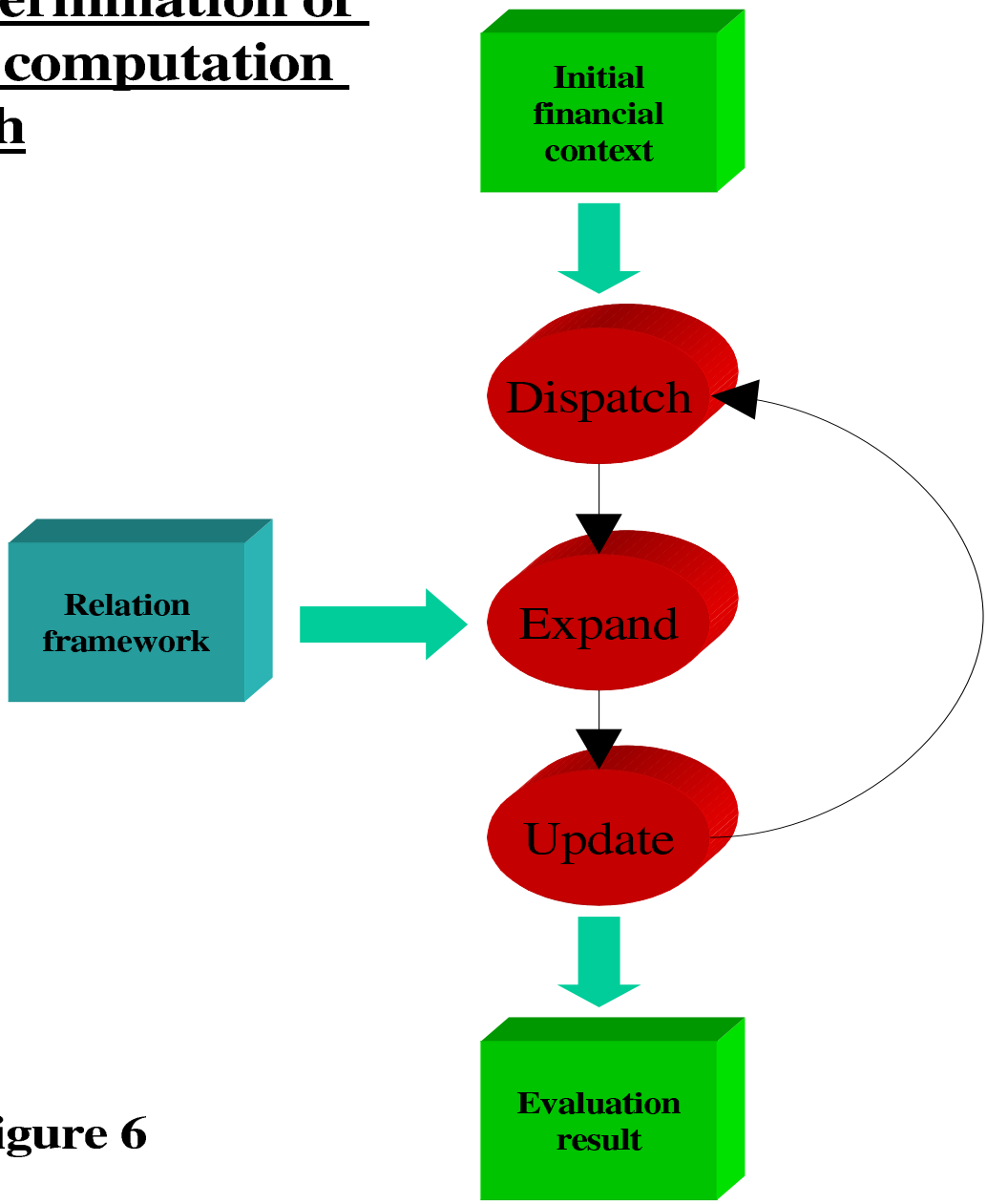
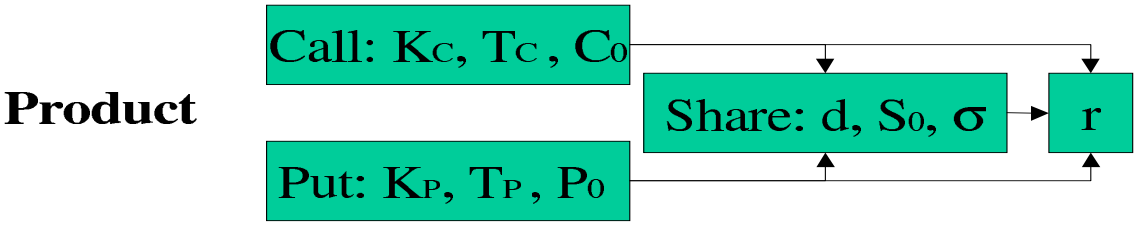
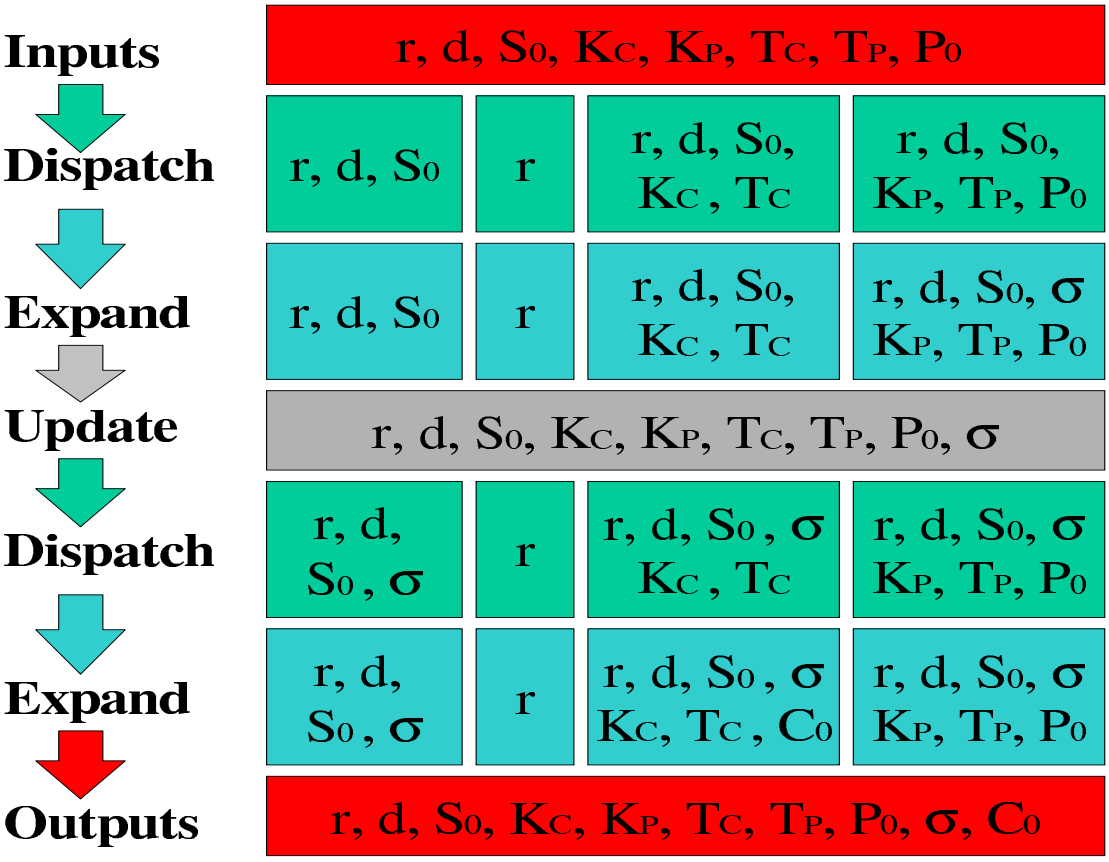


Figure 6



Request: compute the price C_0 of the call

By-product: volatility σ of the share



A naive algorithm is:
• perform the dispatch phase and the expand phase as described above,
• during the updating phase, for every new reachable data, choose a product from which it can be computed and compute its value,
• perform the update phase as described above with respect both to the context base and to the set of values – newly computed values and old values – associated with the context base.

3.3 Eliminating redundant computations

The algorithm just stated generally produces redundant data and the worst case leads to the maximal number of redundancies – largest number of unnecessary computations.
We now describe an algorithm that eliminates redundant computations by performing selective updates:
• perform the dispatch phase as in the dispatch, expand and update algorithm,
• choose a maximal subset of financial products in the portfolio such that their productions are non trivial – i.e. the base of the product is a strict subset of the base of the production – and the images of their production base in the product graph are disjoint,
• perform the update phase as described above with respect both to the products in the maximal subset chosen during the expand phase and to all the other untouched products.
The properties $x \subset P(x)$ and $x \subset y \implies P(x) \subset P(y)$ of production functions ensure that this selective algorithm leads to the same result as the naive algorithm, although usually with a higher number of computation steps but a lower number of total computations.

3.4 Eliminating useless computations

The selective algorithm we have described above eliminates redundant computations, however it does not eliminate useless computations. For example, if the portfolio contains two distinct underlyings and two calls – one written on each underlying – this algorithm may yield the prices of both calls even though the scope has been restricted to the price of only one of the calls.

Let x_n be the intersection of the base of the final context given by the previous selective algorithm and of the search scope and let $c_1, \ldots, c_n$ be the n contexts created during the iterative procedure.
We proceed as follows:

• Let a_n be the subset of x_n of all the element of x_n that cannot be obtained through a trivial production from the elements of c_{n-1},

• Given $a \in a_n$, the selective algorithm yields a unique production P_a that associates the subset $P_a^{-1}(a)$ of c_{n-1} with a subset of c_n containing the element a; we set

$$x_{n-1} = \left(\bigcup_{a \in a_n} P_a^{-1}(a) \right) \cup (x_n - a_n)$$

• working backward, we define the sets $x_{n-2}, \ldots, x_0$ in a similar manner – x_i is obtained from x_{i+1},

• starting from the index $i = 0$, we perform in turn, for each value of i, all the computations associated with the productions P_a where a assumes all the values $a \in x_{i+1}$.

Figure 8 illustrates the principle of this algorithm.

3.5 Choosing a computation path with a utility function

Financial computations are usually time and memory consuming, thus it is natural to introduce a utility function for every computation, inversely proportional to the computation cost. However, other factors may be accounted for in the utility function such as:

• the precision of the pricing scheme used,

• how well a given pricing scheme is suited to the specific needs of its caller,

• the probability that an algorithm may abort and require a human intervention – manual correction.

A given set of utility functions can be viewed as an optimisation framework defined on the pricing framework.

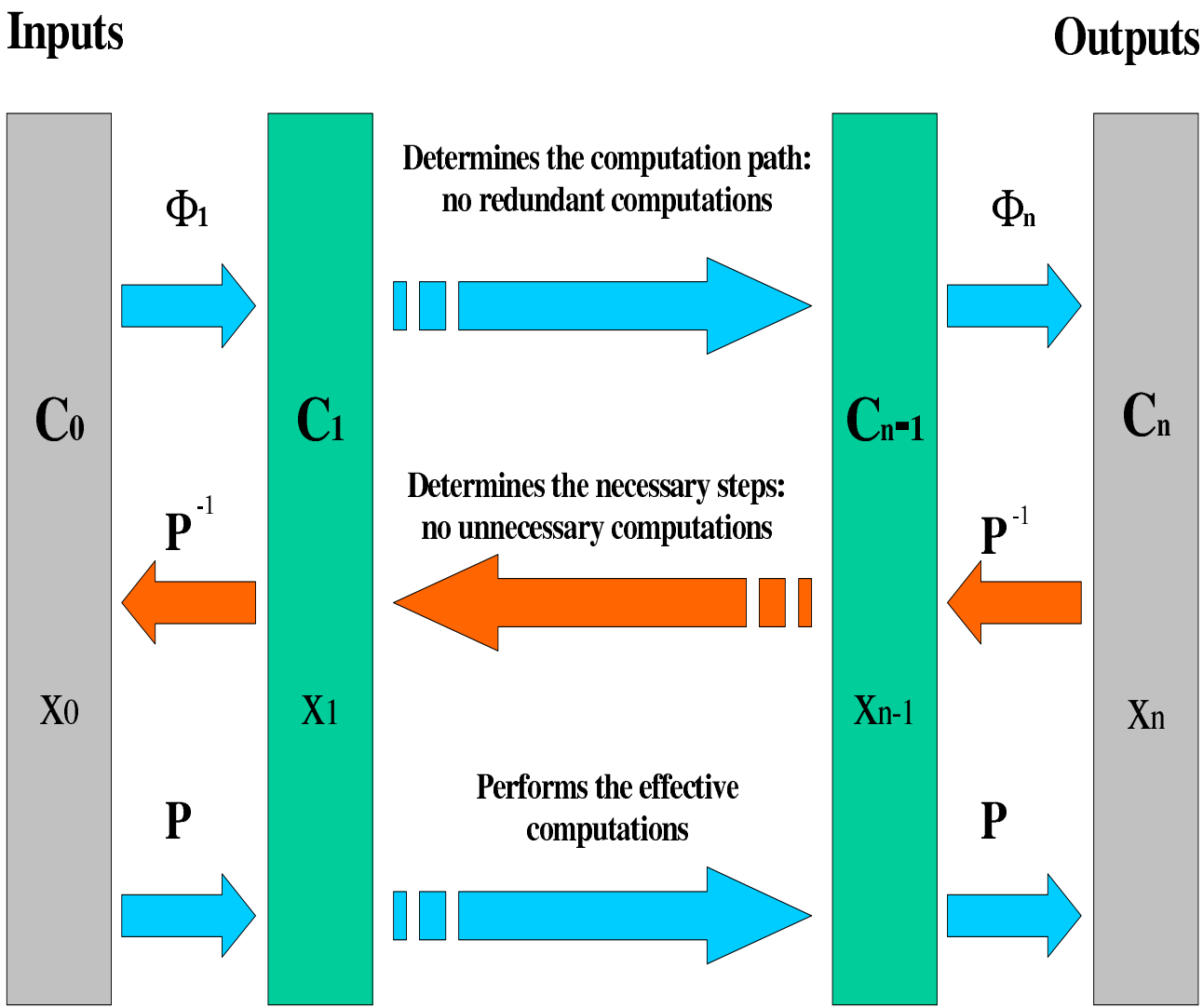

Inputs
Outputs
C0
C1
Cn-1
Cn
Φ1
Φn
Determines the computation path:
no redundant computations
P -1
Determines the necessary steps:
no unnecessary computations
P -1
X0
X1
Xn-1
Xn
P
Performs the effective
computations
P

Figure 8

The selective algorithm described above may be reformulated to account for the optimisation framework:

• the dispatch phase is the same as that of the pure selective algorithm,

• in the update phase, instead of choosing any maximal product subset, we choose a maximal subset A that maximises the sum of its utility functions — i.e.

$$\sum_{a \in A} U_a = \max_{B \text{ maximal}} \sum_{b \in B} U_b$$

Again this algorithm clearly leads to the same result as the naive algorithm and:

• at every computation step, the algorithm maximises a given utility function associated with the optimisation framework,

• it converges in the same number of steps as the simple selective algorithm.

Extensions to parallel computing

The utility function optimisation algorithm described above is a greedy algorithm. Indeed, it chooses the best computation – with respect to a given set of utility criteria – at every time step, and this corresponds to what is expected from a risk management system as the control must be focused on elementary computation steps. Finding a globally optimal computation path is altogether out of the risk management scope and nearly untractable as the algorithm determining the shortest path between two vertices of a graph with n vertices has complexity $O(n^2)$ – remember that here n is of the kind $O(2^p)$ where p is the number of attributes, so this would be 4^p.

Another version of the selective algorithm for the determination of a non-redundant computation path is:

• perform the dispatch phase as for the selective algorithm,

• choose a product such that the production of the projection of the context base on that product is not trivial and perform the expansion with respect to that unique product,

• perform the update phase as for the selective algorithm.

We shall give two reasons why we favour the first selective algorithm we have introduced:

• the algorithm is parallel by its very nature as at every computation step it picks out all the necessary computations that can be performed

simultaneously – the synchronisation process is taken care of in the update phase –,

• the algorithm is much more amenable than its sequential version to local optimisation with respect to a given set of utility criteria.

Obviously, the algorithm can be further parallelized if it can be shown that some productions are disjoint, in which case they would not need to be updated simultaneously. After the determination of the computation path, the knowledge of the relational framework makes it possible to perform this further analysis.

References

Black, F., and M. Scholes, 1973, "The pricing of options and corporate liabilities", Journal of Political Economy, 81., pp. 637-659.

Dordain, J.N., and N. Singh, 1999, *Finance quantitative*, Economica, Paris.

Gamma, E., R. Helm, R. Johnson and J. Vlissides, 1995, *Design patterns*, Addison-Wesley, Reading, MA.

Hull, J., and A. White, 1994, "Numerical procedures for implementing term structure models", Journal of Derivatives, 5., pp. 177-188.

Knuth, D., 1973, *The art of computer programming*, Volume 1, 2 and 3, Addison-Wesley, Reading, MA.

Smithson, C.W., C.W. Smith Jr. and D. S. Wilford, 1995, *Managing financial risk*, Irwin, New-York.

Weiss, D.M., 1993, "After the trade is made: processing securities transactions", New-York Institute of Finance.