Structured Products
and Time Travel

The challenges of valuing structured products on past, present, and future dates

---

Pricing-orientated *payoff* languages form the vast majority of existing financial product language implementations, but face serious limitations for "time travel."

A new class of context-neutral *contract* languages offers key properties to overcome the challenges of valuing structured products on past, present, and future dates.

Technical Brief

There are many instances in which structured products need to be valued on dates other than today.

Consider the following situations:

- A salesman simulates the value of a proposed transaction, six months from today, to verify that the customer's investment guidelines are satisfied.

- A trader calculates the month-to-date P&L on a portfolio of structured products.

- A risk manager calculates six-month VaR on the same portfolio of structured products.

This article describes the challenges of valuing structured products on past, present, and future dates, and reviews potential solutions.

The Challenges of Time Travel

1.   The salesman may first attempt to move the valuation date forward and ignore intermediate paths followed by the contract's underlying variables between today's date and the future valuation date, as illustrated in the "New Contract" example in Figure 1 below. The salesman must also make assumptions about the market conditions that will prevail on the future date. This approach would work, for example, to compare the estimated value of a European option on the future date to the current value of the investor's portfolio.

However, there are many cases in which intermediate paths cannot be ignored:

- The contract may be undefined on the future date, and therefore impossible to value, without prior knowledge of all intermediate fixings. For example, the floating leg of vanilla swap is not defined if a fixing occurs shortly before the future date. An average rate option cannot be fully specified without the value of the running average on the future date. The future composition of the underlying basket in a Himalaya contract cannot be determined without information about all intermediate fixings.

- In addition, certain metrics require the knowledge of intermediate fixings. If the investor's constraint is related to a potential loss on the proposed transaction, the salesman must calculate the contract's fair value at inception and in six months' time, and the value of all intermediate cash flows reinvested (or funded) until the future date, according to a predefined financing policy.

In such cases, the salesman needs to simulate all intermediate paths followed by contract underlyings—as illustrated in Figure 1, in the "Existing Contract" example, in the segment located to the right of the current date—and value the contract on the future date for each simulated path.

Figure 1
Contract Valuation on Past, Present, and Future Dates
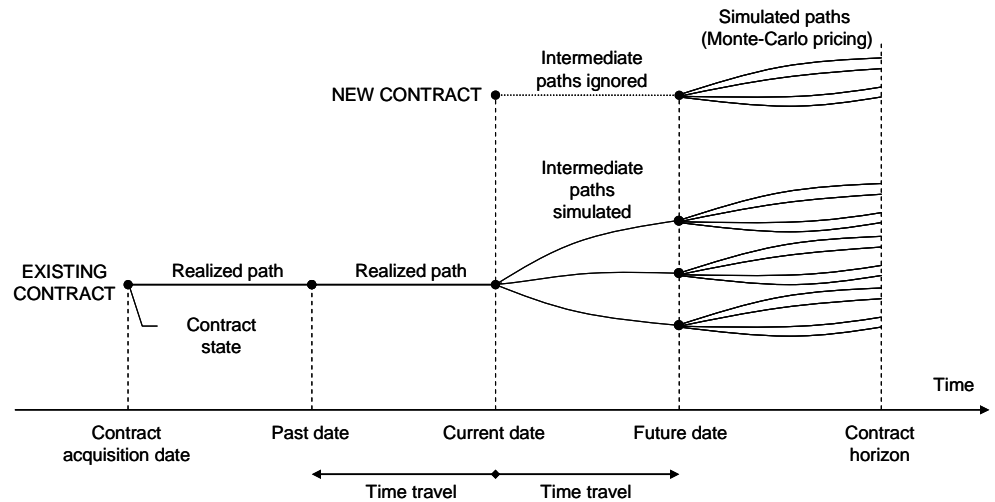
The need to simulate intermediate paths raises additional issues:

- The salesman must monitor potential barrier crossings "continuously"—e.g., daily—throughout the intermediate period.

- The handling of options potentially necessitates full pricing capabilities along the path: for example, if the option gives a choice between receiving one of two complicated sub-contracts and doing nothing, each of the residual sub-contracts needs to be accurately valued on the option's expiry date, taking into account the effect of intermediate paths, to determine the payoff.

- Options with discretionary exercise require additional assumptions about the option holder's behavior. For example, the salesman may define rules based on optimal early exercise conditions (i.e., compare the option's intrinsic and economic values) on each exercise date for Bermudan options and continuously for American options.

The salesman may also make the simplifying assumption that no event occurred in the intermediate period.

2.   Contrary to the salesman, the trader only looks back in time: to calculate a month-to-date P&L, he needs to (i) describe the contract at the end of the previous month and on the current date, taking into account realized events in both cases, (ii) record intermediate cash flows between the past date and the current date, and (iii) define a financing strategy to calculate the portfolio's funding costs since the end of the previous month.

3.   The risk manager needs to represent the contract in its current state, like the trader, and deal with the difficulties of calculating the change in value between today and the future date, like the salesman.

To summarize, a time travel machine for structured products must include the following features:

- *Contract representation.* Accurately specify sophisticated contracts.

- *Scenario*s. Define forward-looking market scenarios.

- *Pricing.* Value contracts at any point in their life cycle.

- *Contract execution.* Define rules that describe how contracts evolve along realized or simulated scenarios, and correctly represent residual contracts in each past, current, and future state.

- *Trading strategies.* Define a financing policy and calculate corresponding costs. A funding policy is like a trading strategy: a list of funding transactions is produced periodically—e.g., daily—based on pre-defined rules and cash flows generated by the portfolio.

The choice of an appropriate formalism for describing financial products is a key decision in the design of a complete valuation system. The remainder of this article reviews potential solutions for representing and valuing structured products throughout their life.

Payoff Language vs. Contract Language

We assume that structured products are modeled with a programming language: a language approach provides the flexibility needed to describe the great diversity of terms and conditions that characterize structured products.

In the past decade, many derivatives desks have developed *payoff* languages that define a product's meaning in the context of pricing. For example, in a Monte-Carlo pricing context, a product is a program—a "machine"—that takes a simulated market scenario as input and returns the list of cash flows induced by the scenario, on the correct dates. Products are mapped from inception into the world of valuation algorithms: cash flows, options, and other potential events are expressed in a way that is understandable by the stochastic machinery used to value financial contracts. This mapping implies a loss of information: for example, a payoff language will not distinguish between a cash-settled option and a physically-settled option, or between an option that is exercised automatically if it expires in-the-money and one that is exercised only at the discretion of one party.

Contract languages describe what contracts "are" independently from what they "do," and the resulting specification is used to derive pricing, simulation, operational management, and other processing capabilities, with the guarantee that the behavior across these environments is consistent. This is in contrast with payoff languages where the meaning of a product is tightly linked to a pricing context. In a contract language, the essence of financial products is described independently from the processes that may be applied to them: contract definitions represent "data" that numerous programs can analyze and translate into equivalent definitions, adapted to users' processing needs. The original product representation is preserved and remains independent from process-specific translations. A contract definition may be translated into a payoff description, but the opposite is not true.

We believe that payoff languages are well suited for the limited task of valuing a new transaction today, while contract languages greatly simplify the task of analyzing both new and existing transactions on past, present, and future dates, where applicable.

The greatest challenge faced by payoff language implementers is to represent the evolving structure of a contract. Payoff language developers have two options:

- *Maintain an external "historical" scenario.*   The information about realized fixings is not stored with the payoff's definition but maintained externally in a historical scenario. The approach presents two potential problems: first, it is not likely to satisfy the requirements of the back office because fixing information is specific to a contract and sometimes differs from the official rates, for example, when the parties negotiate a fixing value in unusual market circumstances. It may be impossible to maintain a single external scenario that reflects such situations for all products. Secondly, discretionary events such as the early exercise of Bermudan and American options cannot be inferred from the historical scenario as they are not automatic but depend on the option holder's decision. Consequently, payoffs are redefined manually after each discretionary exercise decision.

- *Annotate the contract with its event history.*   The other option is to annotate payoff definitions. Annotations include information about past fixings, exercise decisions, barrier crossings, etc. With structured products, such annotations can become very complicated. For example, financial engineers need to maintain an array of Booleans to record Bermudan option exercise decisions. Dozens of annotations may be necessary for more sophisticated products. With a payoff language, financial engineers describe a value process for all possible contract execution paths: the program that inspects payoff annotations for each event type up to the valuation date may become quite complex and difficult to maintain.

A contract language operates differently: a precise, process-independent contract definition is first written to describe the contract exhaustively and then translated to meet the user's processing needs. For example, when pricing code is generated, both cash-settled and physically-settled options are mapped to a `max` operator because they are equivalent from a pricing perspective. But we can still determine whether the option is cash-settled or physically-settled by looking at the contract's definition.

The contract's definition may also be transformed to incorporate information about realized events: instead of simply annotating the initial contract definition, a contract language reflects all known event information in a new, simplified contract. The updated contract definition may then be translated again to produce corresponding valuation code. The combined use of a contract language's operational and valuation semantics ensures that pricing code always mirrors the state of each contract.

The ability to rewrite contracts and to reflect new information as it becomes available also has applications in forward-looking simulation. For example, the user may want to study the impact of a new contract execution rule that stipulates that Bermudan or American options are exercised as soon as the intrinsic value exceeds 90% of the economic value. While the original contract definition is unaffected, the simulated contract is transformed in accordance with the revised execution rule and the simulated market scenario. The contract language programmer only needs to rewrite the execution rule, not the contract's definition.

Payoff languages present a number of limitations for time travel:

- *Inefficient pricing.*   Whether historical fixing information is stored externally or as an annotation of the payoff's definition, the contract's history must be replayed for each Monte-Carlo path until the valuation date is reached. 100,000 Monte-Carlo simulations imply 100,000 runs with the same historical scenario.

  In addition, a payoff language is unlikely to reflect the changing dimension of a valuation problem. Consider the case of a Himalaya contract that pays the average performance on a twelve-stock basket. Each quarter the best performing stock is used to contribute to the average performance and then

removed from the basket. Without skillful, ad-hoc programming, a payoff language implementation will simulate all twelve underlyings until the contract matures in three years.

In contrast, a contract language generates pricing code dynamically as contracts transition from state to state. Redundant Monte-Carlo trajectories for both historical scenarios and irrelevant underlyings are eliminated.

- *Limited flexibility to change models, numerical methods, and contract execution rules.* Because they do not decouple contract definitions and processes, payoff languages offer limited flexibility to accommodate changes in models, numerical implementations, and contract execution rules.

  For example, a valuation system must exploit the performance gains afforded by optimized pricing algorithms and closed forms. More specifically, the system should provide a choice between a default numerical method and optimized pricing algorithms. Both payoff and contract languages offer the ability to recognize contracts that should be priced with an optimized model or a closed form instead of a default numerical method. In a payoff language, calls to custom pricing functions are hard-coded in the product's definition. As a result, users may only use one valuation method for each product definition. Because a contract language treats contract definitions as data and introduces information about custom pricing algorithms in model definitions instead of contract definitions, users have a choice of either using optimized pricing algorithms or calling the full succession of default numerical calculation steps. This feature facilitates the concurrent deployment of simple models, typically used for risk management, and more sophisticated models, used for pricing.

  As we explained above, a contract language also enables the simulation of multiple contract execution rules—e.g., methods for handling Bermudan and American options in a simulation—without any change to the original contract definition, whereas a payoff language would require a separate definition for each contract execution rule.

- *Market conventions in pricing code.* Payoff languages mix product definitions and pricing specifications. As a result, valuation model implementers must deal with market conventions and business rules. With a contract language, all market conventions are resolved in the contract's definition. Model implementers can focus exclusively on the mathematics of valuation.

- *No back office reconciliation.* Payoffs are defined from inception to generate a series of numerical calculation steps. As we explained, payoff languages describe what contracts "do" as opposed to what they "are:" the contract's essence is not specified. For example:

  - Payoff definitions do not specify whether an option is physically-settled or cash-settled, or whether the exercise is automatic or at the discretion of one party.

  - Payoff languages often do not explicitly describe products that generate cash flows in multiple currencies. Instead, payoff definitions are normalized into a single currency: users specify an amount and an exchange rate instead of a foreign currency amount.

  - Likewise, programmers will minimize the number of annotations needed to maintain payoff definitions: for example, in a flexi cap where only a portion of caplets can be exercised, a payoff representation will simply record the number of exercised caplets, while a contract representation will keep the complete description of exercised caplets.

The loss of information resulting from the use of pricing-orientated payoff languages precludes any reconciliation with a corresponding product representation in the back office. A context-neutral representation is required for this task.

- *Cumbersome event planning.* Future contract life cycle events are easier to detect with a contract language than with a payoff language. A contract language treats product definitions as data, which may be analyzed and translated into the desired output—e.g., the list of all future contract events. In addition, a contract language facilitates the analysis of an existing contract, with a realized event history, in order to determine a revised list of future events. In contrast, payoff definitions contain pricing-orientated code as opposed to data. Payoff code does not lend itself to introspection: a comprehensive list of future events can be very difficult, if not impossible, to extract.

- *Risk of double counting.* Payoff language programmers must be careful to exclude cash flows that hit the cash balance from pricing. Contract languages eliminate this important source of programming errors by generating valuation code that exactly mirrors the state of each contract.

Conclusion

The valuation of structured products on past, present, and future dates presents a number of challenges.

A general valuation system for structured products must include (i) a precise representation of contracts, scenarios, and trading strategies, (ii) pricing capabilities, and (iii) the ability execute contracts along real and simulated scenarios.

An important decision in the development of such a valuation system is the selection of a financial product representation formalism. A comparative analysis revealed the superiority of contract languages over payoff languages for "traveling in time."

LexiFi®

THE
STRUCTURED
PRODUCT
SOLUTION

LexiFi provides software and services that enable financial institutions to structure, price, and process complex financial products.

January 2004