Designing Financial Swaps with CLP($\Re$)

Evan Tick

CIS-TR-94-21
November 1994

Abstract

This paper describes how to design and evaluate financial swaps using CLP($\Re$), a constraint logic programming language over the real numbers. We give a methodology for handling both interest rate and currency swaps. A large real-life example is given to illustrate the techniques. The analyzer is useful to swap practitioners by allowing quicker and more flexible experimentation over the design space than is currently possible with spreadsheets.

Department of Computer and Information Science
University of Oregon

Contents

"Les affaires, c'est bien simple, c'est l'argent des autres."
Alexandre Dumas the Younger
La Question d'Argent, 1857

1 Introduction

Swaps are financial instruments that allow two parties to exchange interest payments in perhaps different currencies. A swap is a powerful building block from which exchange networks can be built, resulting in redistribution of economic surpluses and risks. Usually an intermediary (financial institution) designs and implements the swap network, taking a bit of the profit as payment. A key criterion for a swap network to be viable is that no party must bear risk beyond its risk preference. For the intermediary, this usually means *no risk*, i.e., all stochastic factors must "cancel out" at the intermediary (we shall see this later).

This paper describes how to design and evaluate financial swaps using CLP($\Re$), a constraint logic programming languages over the real numbers [5, 2]. Although a similar language would do, e.g., GDCC [1], we chose CLP($\Re$) for its robustness and availability. The proposed method is of interest to swap practitioners by allowing quicker and more flexible experimentation over the design space than can be accomplished with current methods, e.g., spreadsheets. For example, exploiting the ability of constraint languages to solve linear equations for any combination of unknown variables, a swap network can be partially constructed without binding all the input parameters. The system will then return the relationship among the unknowns, e.g., give the relation between two interest rates or long-term exchange rates to guarantee a profit within a certain range for a given entity. Using such a tool encourages flexible experimentation and optimization that is not possible with spreadsheets, where the equations can be "solved" in only a rigid fashion.

This paper is organized as follows. An overview of swaps is given in Section 2. Section 3 reviews CLP($\Re$). Section 4 discusses how the swap analyzer is designed and implemented in CLP($\Re$). A large example, the Kodak swap, is explained in Section 5. Conclusions and future work are summarized in Section 7.

2 Review of Swaps

Hull [3], Shapiro [8], and Macfarlane et al. [7] are just a few of the general expositions about swaps. Practitioner's jargon concerning the many varieties of swaps can be found in Layard-Liesching [6]. The subtle assumptions involved in the zero-sum attributes of the swaps discussed here are clarified by Turnbull [9].

2.1 Interest Rate Swaps

Figure 1 illustrates the simplest interest rate swap wherein the two parties A and B have loans of the same principle amount, P. The type of loan we consider extends over some number of periods t_i for $1 \leq i \leq n$. Payments are made (according to the interest rate) each period t_i followed by a lump-sum payment of the entire principle at the last period t_n.

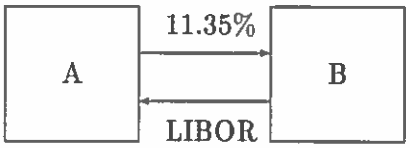


Figure 1: Interest Rate Swap (Building Block)

The swap consists of A making fixed interest payments of 11.35% to B in exchange for receiving floating LIBOR[1] payments from B. For example, a scenario in which this makes sense is when A has a floating-rate loan pegged to LIBOR and B has a fixed-rate loan. We do not show these lenders in the swap network shown in Figure 1. For reasons of risk preference, A wants a fixed rate and B wants a floating rate, and so they swap interest payments.

A swap is effectively a simultaneous exchange of bonds. Using net present valuation, we can compute any of these bond values:

$$B = P - \sum_{i=1}^{n} \frac{s_i P}{(1+r)^{t_i}} - \frac{P}{(1+r)^{t_n}}$$

where s_i is the loan interest rate for period i and r is a fixed market rate. Here we assume that the bonds are risk-free and have the same principle P and length of term n. If we relax our restriction of a fixed market rate, we get:

$$B = P - \sum_{i=1}^{n} \frac{s_i P}{\prod_{k=1}^{i}(1+r_k)} - \frac{P}{\prod_{k=1}^{n}(1+r_k)}$$

where r_k is the market rate for period k. Although the latter formula is implemented in our swap analysis tool, for simplicity we explain swaps using the former equation. Thus for instance we see:

$$\begin{aligned} B_1 &= P - \sum_{i=1}^{n} \frac{0.1135(P)}{(1+r)^{t_i}} - \frac{P}{(1+r)^{t_n}} \\ B_2 &= P - \sum_{i=1}^{n} \frac{\text{LIBOR}_i(P)}{(1+r)^{t_i}} - \frac{P}{(1+r)^{t_n}} \\ \pi_A &= B_1 - B_2 \\ \pi_B &= B_2 - B_1 \end{aligned}$$

Another simplification is to remove the dependence on LIBOR_i for all periods i. Hull [3] briefly discusses how to do this. Effectively, B_2 is the same for any value of $n \geq 1$. Thus

[1] London Interbank Offer Rate — any floating rate will do for our purposes.

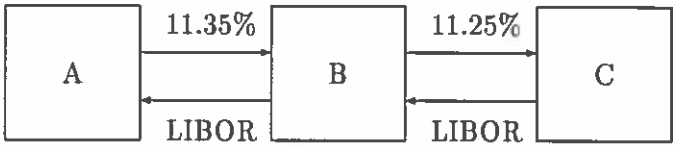


Figure 2: "Plain Vanilla" Interest Rate Swap

pick $n = 1$ to get the simplest relation, based only upon LIBOR_1. Assuming that all parties are risk-free banks, this can be justified by having the initial bondholder pass the bond through to another party after one period. Thus all subsequent cash flows cancel, leaving only the cash flows at the end of the initial period. Effectively the rate floats to ensure that this simplification holds!

From this simple building block we can build more sophisticated networks. Figure 2 shows dual offsetting swaps through an intermediary B. Clearly A effectively transforms a floating to a fixed loan, and C transforms a fixed to a floating loan. B cancels its risk by passing the floating payments through from C to A. If we assume neither A nor C defaults, then B has no risk. In addition, B takes a profit of 0.1% for its service. Valuation gives the additional equations:

$$\begin{aligned} B_3 &= P - \sum_{i=1}^{n} \frac{0.1125(P)}{(1+r)^{t_i}} - \frac{P}{(1+r)^{t_n}} \\ \pi_A &= B_1 - B_2 \\ \pi_B &= B_3 - B_1 \\ \pi_C &= B_2 - B_3 \end{aligned}$$

For this simple example, the profit to B can be computed more directly; however, in complex networks, the general formula is needed. To simplify things, we may elect to assume that the market rate is fixed over the length of the loan. In any case, it is critical for evaluating this formula in CLP($\Re$) that the market rate(s) be known *a priori*, otherwise nonlinear equations arise.

In addition to previous bond-like loans, amortized loans, wherein the principle is incrementally repaid, are easily modeled. Our system supports a library of these various types of loans.

2.2 Currency Swaps

Figure 3 shows a simple currency swap building block. Here parties A and B lend each other principles in yen and dollars, respectively, of approximately the same value. They then pay each other interest based on those principles, until the end of the loan, when the principles are repaid. To alleviate foreign exchange risk at period t_n when the principles are repaid, a forward exchange rate, $F_{\$/Y}$ may be agreed upon in the swap agreement. For

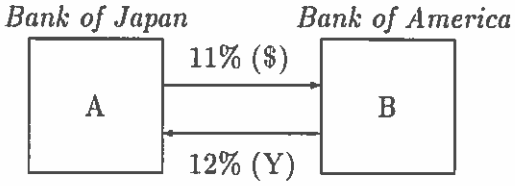


Figure 3: Currency Swap (Building Block)

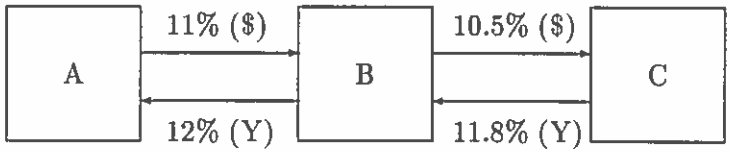


Figure 4: "Plain Deal" Interest Rate Swap

example, suppose the original principles are $P_\$$ and P_Y, where $P_\$ = S_{\$/Y} P_Y$ at t_0 given the spot exchange rate $S_{\$/Y}$. Then at t_n, parties A and B might replace principles $P'_\$$ and P_Y respectively, where $P'_\$ = F_{\$/Y} P_Y$.

The previous bond valuation formulae still hold, where the dollar bond value is B_1 and the yen bond value is B_2:

$$\begin{aligned} \pi_A &= B_1 - B_2 = S_{Y/\$} B_1 - B_2 \\ \pi_B &= B_2 - B_1 = S_{\$/Y} B_2 - B_1 \end{aligned}$$

In the above we compute the current value of the swap to parties A and B in today's yen and dollars respectively.

From this simple building block we can build more sophisticated networks. Figure 4 shows dual offsetting swaps through an intermediary B. Valuation of the swap follows from the previous discussion. A *circus swap*[2] is a combination of plain vanilla interest rate swap and plain deal currency swap [3], i.e., basically the swap of Figure 4 with either currency's loans on a floating rate. We will see an example of this in the larger example discussed in Section 5.

In summary, the building blocks for composing swap networks use elementary cash flow mathematics, which facilitate their expression in CLP($\Re$). The key underlying stochastic variables — floating interest rates, market investment rates, and currency exchange rates — are problematic. Although one could couple our proposed tool with a sophisticated models for predicting these rates, the issue of prediction is orthogonal to that of building and analyzing the swap networks themselves. Therefore we assume in this paper that these

[2] Picadilly or Ringling Brothers?

```
flat( Start, End, _, _, _, 0 ) :- Start >= End.
flat( Start, End, Principle, Rate, MR, Value ) :-
   Start < End,
   Value = Principle - Payments
   loan( End-Start, Principle, Rate/100, MR/100, Payments ).

loan( Time, In, Rate, MR, Value ) :-
   Time > 0, Time <= 1,
   Out = In / ( 1 + MR * Time ),
   Value = Out * ( 1 + Rate * Time ).

loan( Time, In, Rate, MR, Value ) :-
   Time > 1,
   Out = In / ( 1 + MR ),
   Value = Next_Value + ( Out * Rate ),
   loan( Time-1, Out, Rate, MR, Next_Value ).
```

Figure 5: Flat Payment Loan Valuation in CLP($\Re$)

stochastic rates are given over the swap term, either as a fixed value or a vector of varying values.

3 Review of CLP($\Re$)

CLP($\Re$) is constraint logic programming language over the domain of real arithmetic. Programs appear in syntax to be Prolog programs, i.e., data and control structures are the same. The semantics of unification, however, are vastly different. We illustrate the language with a simplified version of a loan of the type previously discussed, shown in Figure 5. This procedure computes the net present value of fixed interest loans with fixed market rates only (see Appendix A for more general valuation procedures). Procedure `flat/6` has the following parameters: the `Start` and `End` periods of the loan, the `Principle`, the fixed loan `Rate`, `MR` (a fixed market rate), and `Value` (the net present value of the loan).

If the length of the loan is not positive, the loan value is zero (`flat/6` clause 1). Otherwise, the loan value is the principle minus the payments, computed by `loan/5`, starting at the next period. Procedure `loan/5` computes the payment value in what can be considered an iterative (or recursive) manner; however, the language lends itself to a more elegant *declarative* semantics. In effect, `loan/5` (and any procedure invocation in general) is true if the equations it engenders are consistent over the domain of the reals. Furthermore, these equations are not necessarily evaluated in any strict order: CLP($\Re$) has an internal equation solver that is transparent to the programmer.

The spawned equations form a recurrence. Each successive value is equal to the next value plus the discounted principle multiplied by the interest rate (`loan/5` clause 2). The

final value (at the final period) also includes payback of the entire principle (`loan/5` clause 1). The final period can be fractional, requiring us to scale the loan and market rates by the remaining time.

Examples of queries to this program are instructive. The value of a three year $100 loan at 10% assuming a 5% market rate is:

```
?- flat( 1, 4, 100, 10, 5, V ).

V = -13.6162
```

Alternatively we can solve for loan rate:

```
?- flat( 1, 4, 100, R, 5, -14 ).

R = 10.14
```

However, we cannot solve for the market rate because the function is nonlinear in this variable. More strangely, we cannot solve for the time. For example, trying to solve for the ending period:

```
?- flat( 1, E, 100, 10, 5, -14 ).

E <= 2
1 < E
114 = _t13 * (0.1*E + 0.9)
100 = (0.05*E + 0.95) * _t13

*** (Maybe) Retry?
```

The "maybe" caveat in the result indicates that the non-linearity could not be removed and that the solution may be inconsistent. We can avoid CLP($\Re$) confusion by simplifying clause 1 of `loan/5` as:

```
loan( Time, In, Rate, MR, Value ) :-
   Time > 0, Time <= 1,
   Value * (1 + MR * Time ) = In * ( 1 + Rate * Time ).
```

There is an art to making such transformations! With this change, the system automatically solves:

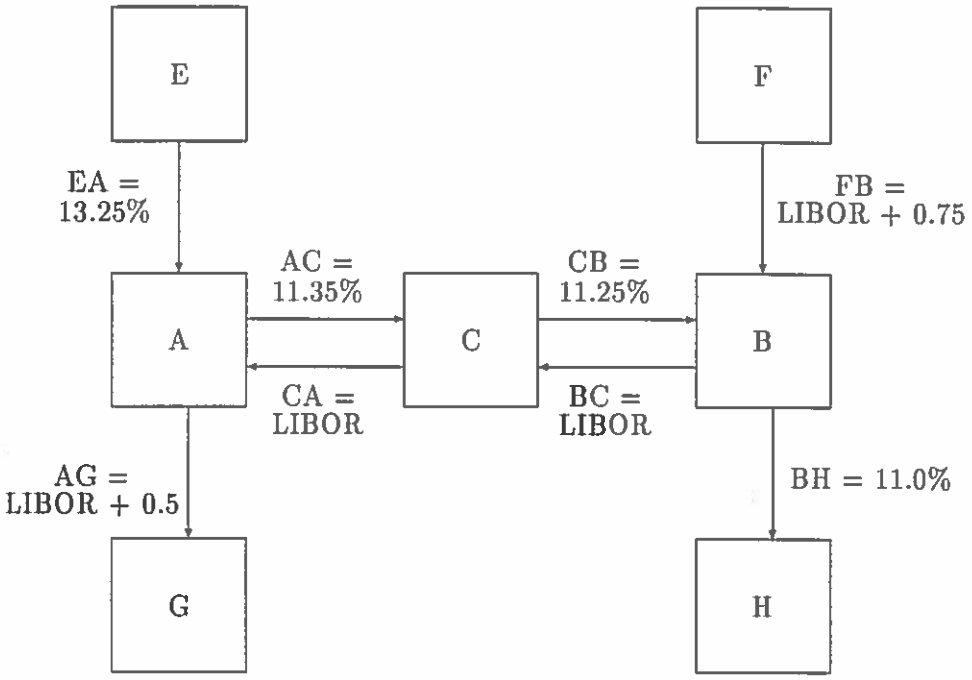
```
?- flat( 1, E, 100, 10, 5, -14 ).

E = 4.55
```

It is important to note that to solve for time, the final fractional scaling of the loan rate is required. Without this, `loan/5` would not be able to ground the recurrence when solving for time (`E`), i.e., it loops forever. Although the analyzer we built uses loan procedures that are more sophisticated that this, the foundation is the same. Additional complexity arises from (optional) variable loan and market rates and amortization.

4 Swap Analysis in CLP($\Re$)

This section describes the design construction of the swap analyzer, as a series of increasingly sophisticated models. Figure 6 shows an interest rate swap and its straightforward




```
net1( Info, [ Pi_A, Pi_B, Pi_C ] ) :-
   Info = [ AC, CA, CB, BC, AG, BH, FB, EA ],
   Pi_A = CA - AC - AG + EA,
   Pi_B = FB - BH - BC + CB,
   Pi_C = AC + BC - CA - CB.
```

Figure 6: Plain Vanilla Interest Rate Swap in CLP($\Re$)

```
net2( [P, R_mkt, T], Info, Libor, [ Pi_A, Pi_B, Pi_C ] ) :-
   Info = [ AC, CA, CB, BC, AG, BH, FB, EA ],
   Pi_A = AC_CF + AG_CF - CA_CF - EA_CF,
   Pi_B = BH_CF + BC_CF - FB_CF - CB_CF,
   Pi_C = CA_CF + CB_CF - AC_CF - BC_CF,
   loan(  P, EA,         R_mkt, T, EA_CF ),
   loan(  P, AC,         R_mkt, T, AC_CF ),
   loan(  P, CB,         R_mkt, T, CB_CF ),
   loan(  P, BH,         R_mkt, T, BH_CF ),
   floan( P, CA, Libor, R_mkt, T, CA_CF ),
   floan( P, AG, Libor, R_mkt, T, AG_CF ),
   floan( P, BC, Libor, R_mkt, T, BC_CF ),
   floan( P, FB, Libor, R_mkt, T, FB_CF ).
```

Figure 7: Cash-Flow Model in CLP($\Re$) for Previous Network

translation into CLP($\Re$) program, where the loan structures are identical except for the rates. Essentially each node in the network corresponds to an equation balancing the interest rates entering/exiting that node. This simple "rate" methodology for evaluating the swap is possible because the loan principles and terms are identical. This model also assumes a fixed market rate. A typical query to this program is:

```
?- net1( [ 11.35, LIBOR, 11.25, LIBOR, LIBOR+0.5, 11.0, LIBOR+0.75, 13.25 ], Pi ).

Pi = [1.4, 1, 0.1]
```

When the swap calls for differing principles or terms, then individual cash flows must be computing using the bond valuation formula. This model is considered in Figure 7, which shows the CLP($\Re$) implementation of a cash-flow model of the previous network. We invoke the `loan/5` and `floan/6` procedures for a fixed principle of \$100M and 5 period loan length. The net present values are computed from the cash flows rather than the interest rates as in Figure 6. Clearly we could give each loan independent principles and lengths if we desired. Interestingly, we need never define `Libor`: it will be instantiated as necessary and shared among the four floating loans. All unknown LIBOR terms will cancel from the solved equations! For example, typical queries to the program include:

```
?- net2( [100,10,5], [ 11.35, 0, 11.25, 0, 0.5, 11.0, 0.75, 13.25 ], _, Pi ).

Pi = [5.3071, 3.79079, 0.379079]

?- net2( [100,10,5], [ X, 0, 11.25, 0, 0.5, 11.0, Y, 13.25 ], _, Pi ).

Pi = [-4.19247*X + 53.454, 4.19247*Y + 1.04812, 4.19247*X - 47.1653]
```

These solutions are in dollars and are consistent with the previous (rate model) solution in terms of interest rates.

```
net3( Info, Libor, [ Pi_A, Pi_B, Pi_C ] ) :-
   R_mkt = 10,
   Info = [ ( Pac, Tac, AC ),
            ( Pca, Tca, CA ),
            ( Pcb, Tcb, CB ),
            ( Pbc, Tbc, BC ),
            ( Pag, Tag, AG ),
            ( Pbh, Tbh, BH ),
            ( Pfb, Tfb, FB ),
            ( Pea, Tea, EA )
          ],
   Pi_A = AC_CF + AG_CF - CA_CF - EA_CF,
   Pi_B = BH_CF + BC_CF - FB_CF - CB_CF,
   Pi_C = CA_CF + CB_CF - AC_CF - BC_CF,
   loan(  Pea, EA,         R_mkt, Tea, EA_CF ),
   loan(  Pac, AC,         R_mkt, Tac, AC_CF ),
   loan(  Pcb, CB,         R_mkt, Tcb, CB_CF ),
   loan(  Pbh, BH,         R_mkt, Tbh, BH_CF ),
   floan( Pca, CA, Libor, R_mkt, Tca, CA_CF ),
   floan( Pag, AG, Libor, R_mkt, Tag, AG_CF ),
   floan( Pbc, BC, Libor, R_mkt, Tbc, BC_CF ),
   floan( Pfb, FB, Libor, R_mkt, Tfb, FB_CF ).
```

Figure 8: Cash-Flow Model in CLP($\Re$) with Extended Parameters

Figure 8 shows an extended implementation of the network with full input parameters allowing each node to have a different principle and loan length. For example, given this procedure, we can query:

```
?- net3( [ (90,6,11.35), (100,4,0), (100,5,11.25), (100,5,0),
           (100,5,0.5), (100,5,11.0), (100,5,0.75), (100,5,13.25) ],
         [L1,L2,L3,L4,L5,L6,L7,L8], Pi ).

Pi = [-62.0921*L5 + 11.3422, 3.79079, 62.0921*L5 - 5.65605]
```

which gives the profits when the fixed loan to A (from C) is extended to 6 periods on a reduced principle of 90 and the floating loan to C (from A) is reduced to 4 periods. Note that the profit to B has not changed. The change in the profits to A and C are a function of the parameter L5 which is the last period LIBOR rate. The previous four LIBOR rates still cancel by the swap. As the loan lengths become more disparate, we expect to see more LIBOR rate terms in the solutions.

This final model is similar to our prototype. Additional facilities include fixed amortized loans, principles in alternative currencies, and forward exchanges. The market rate is implemented in a manner similar to LIBOR. A fundamental difference is that in the analyzer we built, programs such as `net3` are generated *automatically* from net specifications with are in turn generated from graphical input supplied by the user. Thus one crucial design

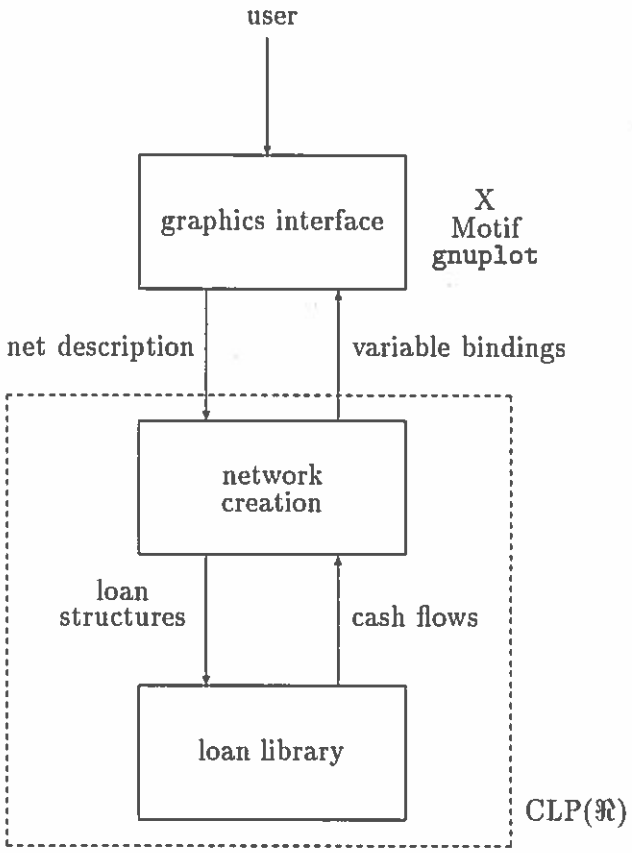


Figure 9: Overview of Swap Analysis System

philosophy we adopted was to shelter the user from CLP($\Re$). This impacted the flexibility with which the system can be used, as is discussed below.

Figure 9 shows the system overview of our current prototype analyzer. The user interface, written in C, accepts graphical entry of the network and translates it into a net description accepted by the analyzer, written in CLP($\Re$). See Appendix A for the analyzer source listing.

The net description can have symbolic names for parameters, which if bound are returned as solutions. In addition, a profit is computed and returned for each node in the graph, which is the sum of its cash flows. Internal to the analyzer itself, the net description is used to invoke loan library routines that define various types of payments, such as simple or amortized. These invocations return the cash flow values needed to compute the net present value profits.

The user interface is illustrated in Figure 10. The interface allows the user to graphically specify the swap network, entering parameters for each entity (node) and loan (edge). Either real values or symbolic names can be assigned to parameters. A sketch of the information is displayed on the illustrated graph, with detailed information available by explicit querying the interface (with a mouse). The user can also specify constraints in terms of both symbolic input parameters as well as profits. The use of this facility is illustrated in Section 5.

The interface also translates variable bindings into graphics in the limited cases when the binding is an equation in one or two independent variables. We generate a nonparameterized graph description for `gnuplot`. This is also illustrated in Section 5.

4.1 Symbolic Output

Whenever expressing symbolic solutions in CLP($\Re$), the issue of which symbolic variables in the formula are dependent and which are independent looms large. Flexibility in controlling the relative independence of variables is achieved with the `dump/3` predicate [2]. `dump/3` takes a list of variables as input, where the variables earlier in the list order are more independent than later variables. `dump/3` displays dependent variables in terms of independent variables specified by this list. If we purposely remove certain independent variables from the list, we can receive symbolic answers among the dependent variables.

Because of the great flexibility of output control, it becomes difficult for the analyzer to make autonomous decisions concerning symbolic output construction. Sometimes a user may wish to see a certain relationship among variables that would not abide by any default we could provide. Therefore, in the user interface we provide the ability for the user to specify the `dump/3` control list explicitly. A default is presented:

```
[ L1, L2, ..., M1, M2, ..., U1, U2, ..., Pi1, Pi2, ... ]
```

where `L1` is the LIBOR rate in period 1, `M1` is the market rate in period 1, `U1` is a user-defined variable, and `Pi1` is the profit of node #1, etc. Any of these may be absent if inappropriate to the problem at hand, e.g., the market rate may be a given constant. By rearranging this list (usually by variable *type*), the user can produce any relationships needed. For example,

```
[ U1, U2, ..., Pi1, Pi2, ... ]
```

would show the profits in terms of the user-defined variables and not the LIBOR rates. Another example is:

```
[ M, Pi1, Pi2 ]
```

might plot each of the two profits as a function of a fixed market rate, whereas

```
[ Pi1, Pi2, M ]
```

might plot the fixed market rate as a function of the two profits. By affecting the formulae produced by CLP($\Re$), this control list indirectly affects graphs produced by `gnuplot` because independent variables are plotted along the X and Y axis.

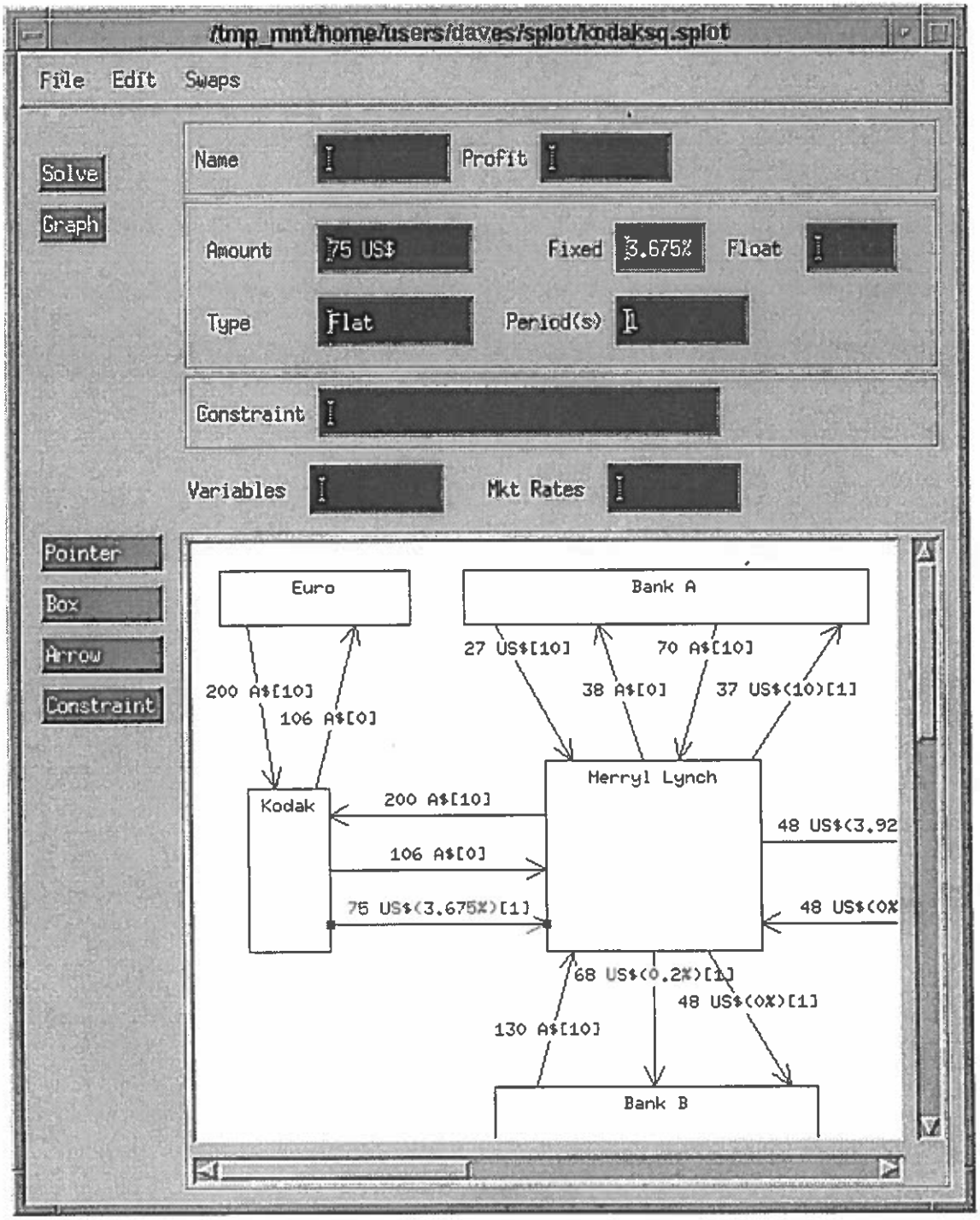


Figure 10: Screen Dump of User Interface to Swap Analyzer

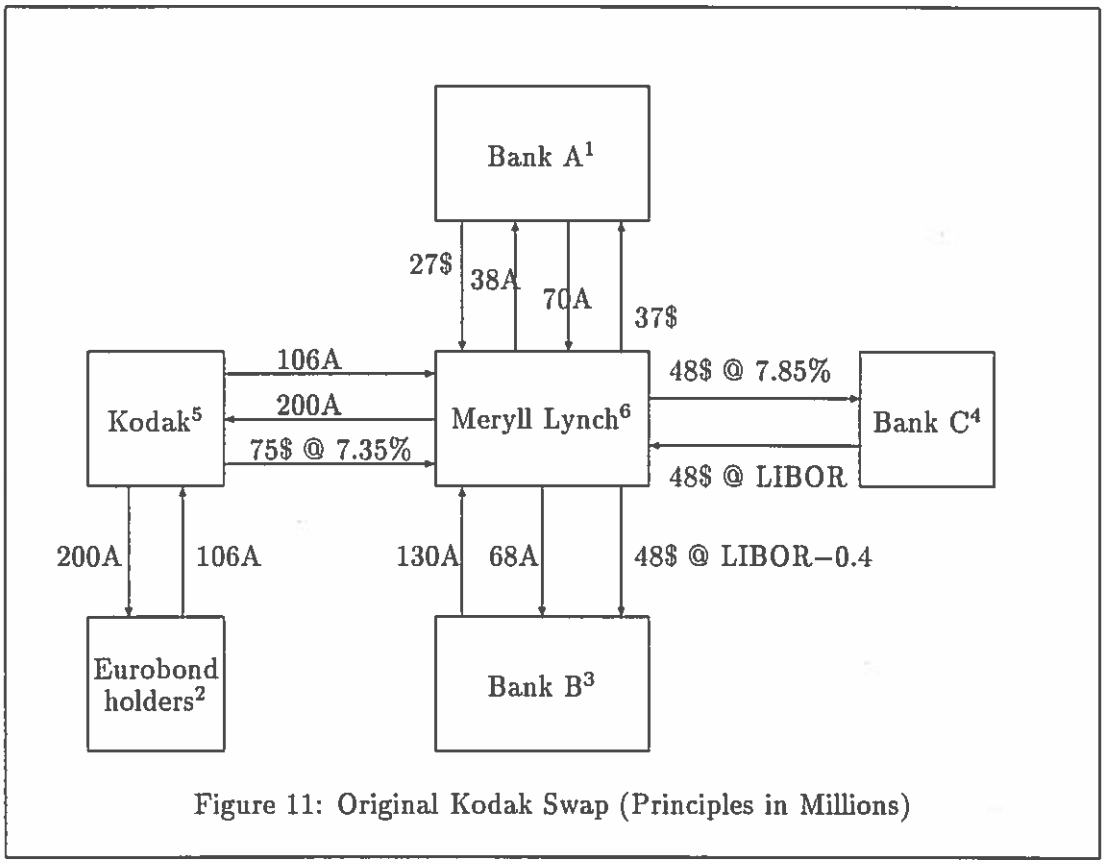


Figure 11: Original Kodak Swap (Principles in Millions)

5 Kodak Example

The Kodak swap [8] illustrates the complexity of swaps in practice, involving two currencies, three banks, an intermediary (Meryll Lynch), and a firm (Kodak). Without going into the detail of the swap agreement, we illustrate the original terms of the swap in Figure 11. Each financial entity is given its own node in the graph, labeled by a node identifier. Edges are annotated with principle amounts (in millions). This figure does not show the implicit five year structure of all loans, nor does it explicitly specify the periods when the currency exchanges are made (when using our swap analyzer, such information must be entered). The swap analyzer can solve this version of the problem, where the result is

$$\begin{aligned} \pi_1 &= -\$9.49 \\ \pi_2 &= \$26.8 \\ \pi_4 &= -\pi_3 - \$17.2 \\ \pi_5 &= -\$1.09 \\ \pi_6 &= \$0.963 \end{aligned}$$

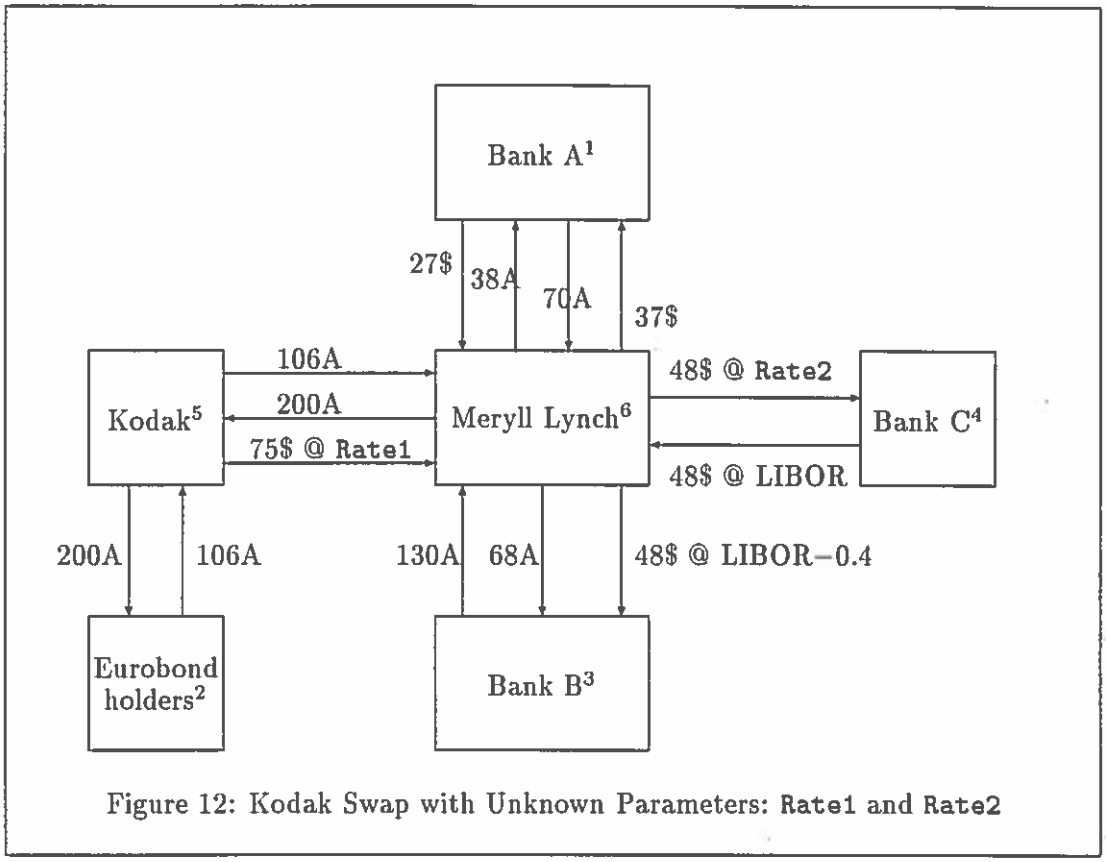


Figure 12: Kodak Swap with Unknown Parameters: `Rate1` and `Rate2`

for U.S. dollar amounts in millions, assuming a spot exchange rate of \$1/0.75A. If we wanted the value of π_3 (or π_4) in detail, we would use an output control specification with LIBOR rates as the most independent variables, giving the following:

$$\begin{aligned}\pi_3 \;=\;& 0.34\mathtt{L10} + 0.35\mathtt{L9} + 0.36\mathtt{L8} + 0.38\mathtt{L7} + 0.39\mathtt{L6} + \\ & 0.40\mathtt{L5} + 0.42\mathtt{L4} + 0.43\mathtt{L3} + 0.45\mathtt{L2} + 0.46\mathtt{L1} - \$32.9\end{aligned}$$

Figure 12 shows the same network with two unknown parameters specified for fixed rates. A typical use would be to view the profits as functions of these parameters. The internal solution produced by the system is:

$$\begin{aligned}\pi_1 &= -\$9.49 \\ \pi_2 &= \$26.8 \\ \pi_4 &= -\pi_3 + 3.99 * \mathtt{Rate2} - 32.9 \\ \pi_5 &= -(6.24 * \mathtt{Rate1}) + 21.8 \\ \pi_6 &= 6.24 * \mathtt{Rate1} - 3.99 * \mathtt{Rate2} - 6.29\end{aligned}$$

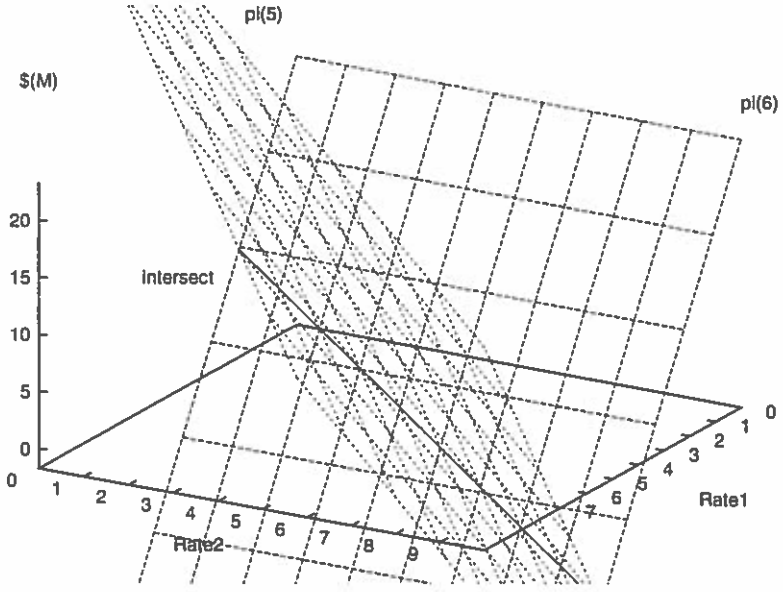


Figure 13: Displayed 3-D **gnuplot** of Profits π_5 and π_6

Any solution (left-hand side variable) that is a function of one or two independent (right-hand side) variables is displayed to the user via **gnuplot**. For example, profits π_5 and π_6 are shown in Figure 13. Making the plots is more difficult than it may look and further research is needed.[3]

A further user facility is the incorporation of constraints in these and other (profit) parameters. Suppose the user specifies that the profits of Meryll Lynch and Kodak are to be equal, by entering the constraint $\pi_5 = \pi_6$. The system can then simplify the solution:

$$\begin{aligned} \pi_1 &= -\$9.49 \\ \pi_2 &= \$26.8 \\ \pi_4 &= -\pi_3 + 3.99 * \texttt{Rate2} - 32.9 \\ \pi_5 &= \pi_6 = -2.0 * \texttt{Rate2} + 7.77 \\ \texttt{Rate1} &= 0.32 * \texttt{Rate2} + 2.25 \end{aligned}$$

which is displayed as in Figure 14. Note that π_4 is a function of π_3 because both are depen-

[3]The actual plots are in color! The intersection of the planes is a parametric equation, and thus the planes must all be parametric for **gnuplot** to display them together. This was done in Figure 13 "by hand." Our current system produces only nonparametric equations, so we cannot compute and display plane intersections (yet). Finding a good vantage point and proper scaling of axes also remain unsolved problems. Currently we rely on **gnuplot** defaults.

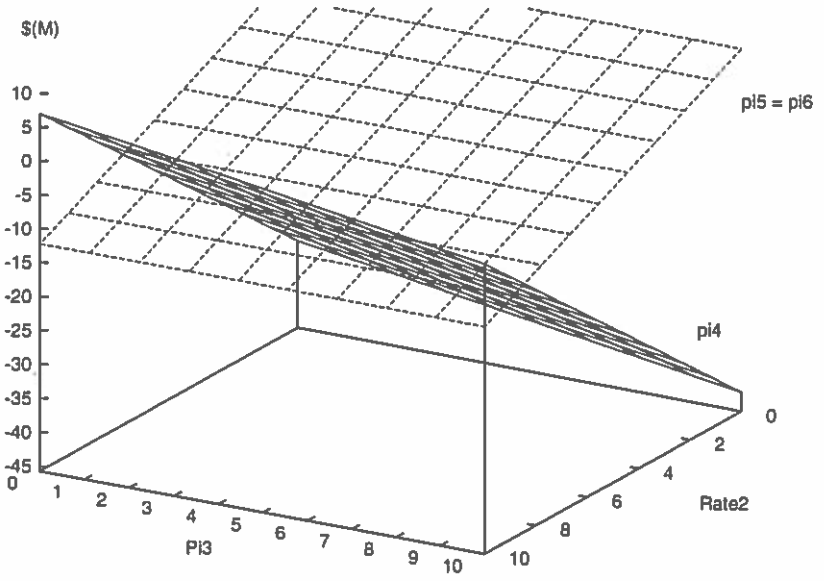


Figure 14: Displayed 2-D gnuplot of Profits π_5 and π_6

dent on LIBOR rates (which Meryll Lynch passes through, so π_6 has no such dependency). The relationship between `Rate1` and `Rate2` to guarantee equal profits is shown above.

If we add the constraint $\pi_3 = \pi_4$, CLP($\Re$) gives us the solution:

$$\begin{aligned} \pi_1 &= -\$9.49 \\ \pi_2 &= \$26.8 \\ \pi_4 &= \pi_3 = 2.0 * \texttt{Rate2} - 16.4 \\ \pi_5 &= \pi_6 = -2.0 * \texttt{Rate2} + 7.77 \end{aligned}$$

plotted in Figure 15. Note that because π_3 and π_4 depend on floating rates in different ways (Bank B receives LIBOR whereas Bank C pays LIBOR), the only way to ensure that the profits are equal is to set `Rate2` as a function of LIBOR itself. The above equations disguise this as π_3 as a function of `Rate2`. However, if we modify the output control specification as:

```
[ L1, L2, L3, L4, L5, L6, L7, L8, L9, L10, Rate1, Rate2 ]
```

we get the direct relationship:

$$\begin{aligned} \texttt{Rate2} = \; & 0.17\texttt{L10} + 0.18\texttt{L9} + 0.18\texttt{L8} + 0.19\texttt{L7} + 0.20\texttt{L6} + \\ & 0.20\texttt{L5} + 0.21\texttt{L4} + 0.22\texttt{L3} + 0.22\texttt{L2} + 0.23\texttt{L1} - \$8.24 \end{aligned}$$

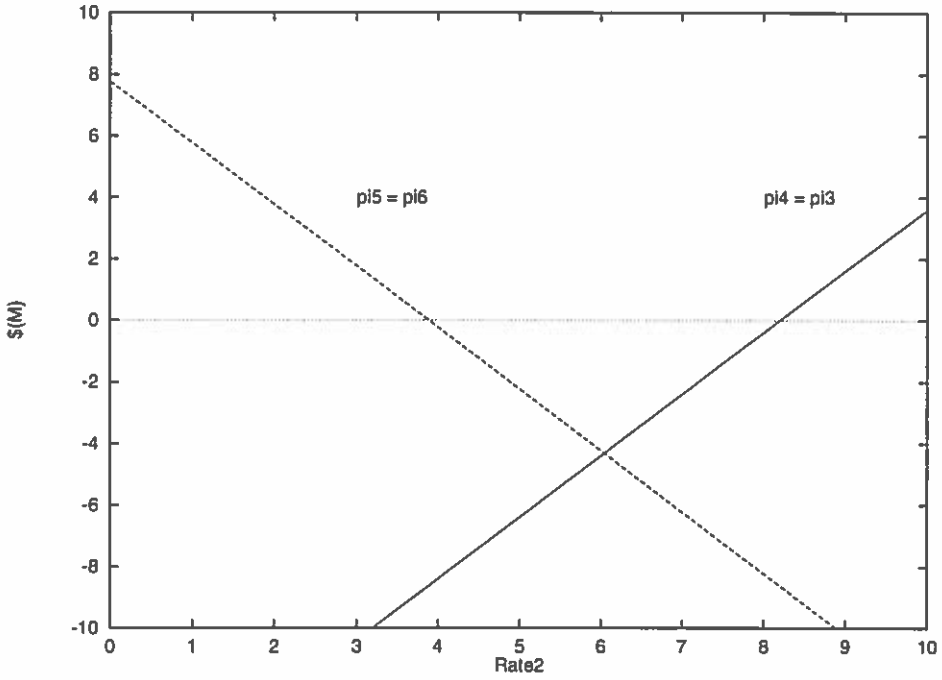


Figure 15: Displayed 2-D `gnuplot` of Profits π_3, π_4, π_5 and π_6

From Figure 15 we see that a value for `Rate2` exists allowing the four profits to be equal. We could solve for this value directly by adding the constraint $\pi_3 = \pi_5$ to the system, getting:

$$\begin{aligned} \pi_1 &= -\$9.49 \\ \pi_2 &= \$26.8 \\ \pi_3 &= \pi_4 = \pi_5 = \pi_6 = -\$4.34 \\ \texttt{Rate1} &= 4.20 \\ \texttt{Rate2} &= 6.66 \end{aligned}$$

meaning that they all lose money. The subtle danger with this solution is that it implies a constraint on LIBOR that may be unrealistic. A mechanism for testing such an over-constrained floating rate is to set the output control specification to:

```
[ L1, L2, L3, L4, L5, L6, L7, L8, L9, L10 ]
```

If any constraints result then the system is over constrained. An empty output indicates a solution with no constraints and so everything is ok. A similar method can be used for testing market rates. We do not yet automatically perform these checks within our system, but it is relatively straightforward to do so.

6 Related Work

A related work in the field of financial engineering is OTAS (Options Trading Analysis System) designed by C. Lassez *et al.* [4] at IBM Yorktown Heights. This system, also based on CLP($\Re$), evaluates the Black-Scholes solution to the partial differential equation describing an option's fair price. The arithmetic involved is more computationally intensive than that of swaps. Because the essential formula is non-linear in the volatility parameter, they use the Steffenson iterative approximation to a linear form allowing solution for volatility.

7 Conclusions

We described a financial swap analysis tool that can accept a high-level description of a swap network and produce functional relationships between unknown parameters, including the net present value (NPV) profits of each entity in the system. The engine of the tool was built in CLP($\Re$), exploiting its ability to perform symbolic arithmetic over the reals, and the user interface was built in C/Motif. The advantage of such a tool is the ability to quickly and flexibly design and evaluate swaps under incomplete information. Profits and parameters can be symbolically constrained to reduce the search space and symbolic solutions can be graphically displayed to help users gain intuition about parametric relationships. These attributes make the tool superior to current analysis methods, specifically trial-and-error evaluation of alternative designs with spreadsheets.

Future work includes:

- gaining more experience with the user interface to determine if further flexibility is required. Especially critical is determination of the best way to control dependent and independent variables, graphical display of solutions, and signaling over-constraint of floating and market rates.
- including a linear approximation routine for evaluating swaps in terms of unknown fixed market rates.
- automating the comparison of swaps evaluated for different market rate structures, to assess risk.

Acknowledgements

This research was supported by an NSF Presidential Young Investigator award, with matching funds from Sequent Computer Systems Inc., and a grant from the Institute for New Generation Computer Technology (ICOT). Raul Clouse helped build the first prototype analyzer and David Scott built the user interface. I thank Bart Massey for many helpful discussions. Peter Stuckey and Roland Yap patiently explained the subtleties of CLP($\Re$) to me.

References

[1] A. Aiba and F. Hasegawa. Constraint Logic Programming System — CAL, GDCC, and Their Constraint Solvers. In *International Conference on Fifth Generation Computer Systems*, pages 113–131, Tokyo, June 1992. ICOT.

[2] N. C. Heintz, J. Jaffar, S. Michaylov, P. J. Stuckey, and R. H. C. Yap. The CLP(R) Programmer's Manual Version 1.2, September 1992.

[3] J. Hull. *Options, Futures and Other Derivative Securities.* Prentice Hall, 1989.

[4] T. Huynh and C. Lassez. An Expert Decision-Support System for Option-Based Investment Strategies. *Computers Mathematical Applications*, 20(9/10):1–14, 1990.

[5] J. Jaffar and J.-L. Lassez. Constraint Logic Programming. In *SIGPLAN Symposium on Principles of Programming Languages*, Munich, 1987. ACM Press.

[6] R. Layard-Liesching. Swap Fever. *Euromoney*, pages 108–113, January 1986. Supplement.

[7] J. Macfarlane, D. R. Ross, and J. Showers. The Interest Rate Swap Market: Yield Mathematics, Terminology and Conventions. Salomon Brothers Inc., June 1985.

[8] Shapiro. *Multinational Financial Management.* Allyn and Bacon, 4th edition, 1992.

[9] S. M. Turnbull. Swaps: A Zero Sum Game? *Financial Management*, 16(1):15–21, Spring 1987.

A Appendix: Source Program

```
/*-------------------------------------------------------------------------
Program: Swap (CLP(R))
Author:  E. Tick
Date:    September 15 1994
Notes:

1. To query:

    ?- top2( +Id, +Exchange, -Out ).

       +Id = identification number of swap data

       +Exchange = 'no' if no currency exchange desired (in final answers)
                   yes( Exchange_List ) if exchange desired (see table/1)

       -Out = solution (list of profits)

    ?- top1( +Id, -Out ).

       same as above, except used default exchange table...
-------------------------------------------------------------------------*/
% mode(?,^)
top1( Id, Out ) :-
   table( FX ),
   top2( Id, FX, Out ).

% mode(?,?,^)
top2( Id, Exchange, Out ) :-
   info( Id,
         info( Market,
               Libor,
               const( Constraints ),
               profits( Profs ),
               vars( Vars ),
               atoms( Atoms ),
               graph( Graph ) ) ),
   market( Market, Mkt ),
   metamaker( Constraints ),                % spawn constraints
   spawn( Graph, Libor, Mkt, Profits ),     % spawn loans
   filter( Exchange, Profits, NewProfits ), % exchange profits
   profit( NewProfits, Out0 ),              % combine profits
   profnamer( Profs, Out0 ),                % bind profits to vars
   !, unconstrain( Libor ),                 % ensure that Libor unbound
   dump( Vars, Atoms, Out ).                % dump vars

% mode(?,?,^)
% no currency exchange for profits...
filter( no, Out, Out ).

% exchange all profit currencies into U.S. $...
filter( yes( Table ), In, Out ) :-
   filter1( In, Table, Out ).
```

```
% mode(?,?,^)
filter1( [], _, [] ).
filter1( [ In | Ins ], Table, Out ) :-
   In = prof( Node1, Node2, Value, Currency ),
   lookup( Table, Currency, Rate ),
   Out = [ prof( Node1, Node2, Value*Rate, usd ) | Outs ],
   filter1( Ins, Table, Outs ).

% mode(?,?,^)
lookup( [], _, 1 ).
lookup( [ fx( Currency, Rate ) | _ ], Currency, Rate ) :- !.
lookup( [ _ | Rest ], Currency, Rate ) :-
   lookup( Rest, Currency, Rate ).

% mode(?,^)
revolve( In, Out ) :-
   revolve1( In, In, Out ).

% mode(?,?,^)
revolve1( [], _, [] ).
revolve1( [ _ | Ins ], In, [ In | Outs ] ) :-
   In = [ A | As ],
   append( As, [ A ], NewIn ),
   revolve1( Ins, NewIn, Outs ).

unconstrain( Libor ) :-
   functor( Libor, libor, Arity ),
   unconstrain1( Arity, Libor ).

unconstrain1( 0, _ ).
unconstrain1( K, Libor ) :- K > 0,
   arg( K, Libor, X ),
   var( X ),
   unconstrain1( K-1, Libor ).

% profnamer/2 ensures that user-defined variable names
% for profits are unified with their corresponding program
% variables.  This is critical for user constraints to work.
%
% mode(?,?)
profnamer( [], _ ).
profnamer( [ Profit | Profits ], Out ) :-
   profmem( Profit, Out ),
   profnamer( Profits, Out ).

% mode(?,?)
profmem( _, [] ) :- !.
profmem( P, [ P | _ ] ) :- !.
profmem( P, [ _ | Qs ] ) :-
   profmem( P, Qs ).

% mode(?,^)
market( market(X), Term ) :- !,
   functor( Term, market, 100 ),
   fill( 100, Term, X ).
```

```
market( Market, Market ).

fill( 0, _, _ ) :- !.
fill( K, T, E ) :- K > 0,
   arg( K, T, E ),
   fill( K-1, T, E ).

% mode(?)
metamaker( [] ).
metamaker( [ C | Cs ] ) :-
   call( C ),
   metamaker( Cs ).

% mode(?,?,?,^)
spawn( [], _, _, [] ).
spawn( [ First | NewInfo ], Libor, Mkt, Profit ) :-
   First = info( Node1, Node2, Type, Cur, Princ, Start, End, Rate ),
   node( Type, Node1, Node2, Cur, Princ,
         Start, End, Rate, Libor, Mkt, [ Prof1, Prof2 ] ),
   Profit = [ Prof1, Prof2 | NewProfit ],
   spawn( NewInfo, Libor, Mkt, NewProfit ).

% mode(?,?,?,?,?,?,?,?,?,?,^)
node( one, Node1, Node2, Cur, Princ, Start, End, _, _, Mkt, Prof ) :-
   one( Princ, Start, End, Mkt, Cash ),
   pack( Node1, Node2, Cash, Cur, Prof ).

node( flat, Node1, Node2, Cur, Princ, Start, End, Rate, Libors, Mkt, Prof ) :-
   flat( Princ, Rate, Start, End, Libors, Mkt, Cash ),
   pack( Node1, Node2, Cash, Cur, Prof ).

node( amort, Node1, Node2, Cur, Princ, Start, End, Rate, _, Mkt, Prof ) :-
   amort( Princ, Start, End, Rate, Mkt, Cash ),
   pack( Node1, Node2, Cash, Cur, Prof ).

pack( Node1, Node2, Cash, Cur, [Prof1, Prof2] ) :-
   Prof1 = prof(Node1, Node2, (-Cash), Cur),
   Prof2 = prof(Node2, Node1, Cash, Cur).

% mode(?,^)
profit( In, Out ) :-
   profit1( In, [], Out ).

% mode(?,?,^)
profit1( [], Out, Out ).
profit1( [ Profit | Profits ], Stack, Out ) :-
   prepper( Profit, Stack, NextStack ),
   profit1( Profits, NextStack, Out ).

% mode(?,?,^)
prepper( prof( N, _, Cash, Currency ), Stack, NextStack ) :-
   member( prof( N, _, Currency ), Stack, Ans, Rest ),
   ( Ans = no ->
     NextStack = [ prof( N, Cash, Currency ) | Stack ]
   ;
```

```
      Ans = prof( A, B, Currency ),
      NewCash = Cash + B,
      NextStack = [ prof( A, NewCash, Currency ) | Rest ]
   ).

% mode(?,?,^,^)
member( prof(A,_,E), Stack, prof(A,B,C), Out ) :-
   Stack = [ prof(A,B,C) | Back ],
   C = E,
   Out = Back.

member( prof(A,_,E), Stack, Rest, Out ) :-
   Stack = [prof(B,C,D) | Back ],
   not( A = B ),
   Out = [ prof(B,C,D) | More ],
   member( prof(A,_,E), Back, Rest, More ).

member( prof(A,_,E), Stack, Rest, Out ) :-
   Stack = [ prof(A,C,D) | Back ],
   not( D = E ),
   Out = [ prof(A,C,D) | More ],
   member( prof(A,_,E), Back, Rest, More ).

member( _, [], no, [] ).

%=====================================================================
% Following library for variable market rate...
% WARNING: cannot solve for Market!

one( Principle, Start, End, Market, Value ) :-
   var( Start ), !,
   one2( End-Start, End, Market, Principle, Value ).

one( Value, Start, End, _, Value ) :-
   Start >= End.

one( Principle, Start, End, Market, Value ) :-
   Start < End,
   one1( End-Start, Start+1, Market, Principle, Value ).

% fractional market rate at ENDING period...
one1( Time, Period, Market, In, Value ) :-
   Time > 0, Time <= 1,
   arg( Period, Market, MR ),
   Value = In / ( 1 + ( ( MR * Time ) / 100 ) ).

one1( Time, Period, Market, In, Value ) :-
   Time > 1,
   arg( Period, Market, MR ),
   Out = In / ( 1 + ( MR / 100 ) ),
   one1( Time-1, Period+1, Market, Out, Value ).

% fractional market rate at STARTING period...
one2( Time, Period, Market, In, Value ) :-
   Time > 0, Time <= 1,
```

```
   arg( Period, Market, MR ),
   Value = In / ( 1 + ( ( MR * Time ) / 100 ) ).

one2( Time, Period, Market, In, Value ) :-
   Time > 1,
   arg( Period, Market, MR ),
   Out = In / ( 1 + ( MR / 100 ) ),
   one2( Time-1, Period-1, Market, Out, Value ).

%-----------------------------------------------------------------
%
%       P        -P*LR      -P*LR    -P(1+LR)
%       |          |          |          |
%       +----------+----------+----------+
%       |          |          |          |
%     Start                              End
%      L(1)       L(2)       L(3)      L(4)
%     MR(1)      MR(2)      MR(3)     MR(4)
%

flat( Principle, Rate, Start, End, Libors, Mkt, Value ) :-
   var(Start), !,
   Value = Principle - Payments,
   flat2( End-Start, End, Principle, Rate, Libors, Mkt, Payments ).

flat( _, _, Start, End, _, _, 0 ) :- Start >= End.

flat( Principle, Rate, Start, End, Libors, Mkt, Value ) :-
   Start < End,
   Value = Principle - Payments,
   flat1( End-Start, Start+1, Principle, Rate, Libors, Mkt, Payments ).

flat1( Time, Period, In, Rate, Libors, Mkt, Value ) :-
   Time > 0, Time <= 1,
   rate( Rate, Libors, Period, LR ),
   arg( Period, Mkt, MR ),
%  approx. final market rate as full period...
   Value = In * ( 1 + LR * Time ) / ( 1 + MR/100 ).

flat1( Time, Period, In, Rate, Libors, Mkt, Value ) :-
   Time > 1,
   rate( Rate, Libors, Period, LR ),
   arg( Period, Mkt, MR ),
   Out = In / (1 + MR/100),
   Value = ( Out * LR ) + Rest,
   flat1( Time-1, Period+1, Out, Rate, Libors, Mkt, Rest ).

flat2( Time, Period, In, Rate, Libors, Mkt, Value ) :-
   Time > 0, Time <= 1,
   rate( Rate, Libors, Period, LR ),
   arg( Period, Mkt, MR ),
%  approx. final market rate as full period...
   Out = In / (1 + MR/100),
%  scale final loan rate at fractional period...
   Value = Out * ( 1 + LR * Time ).
```

```
flat2( Time, Period, In, Rate, Libors, Mkt, Value ) :-
   Time > 1,
   rate( Rate, Libors, Period, LR ),
   arg( Period, Mkt, MR ),
   Out = In / (1 + MR/100),
   Value = ( Out * 1 + LR ) + Rest,
   flat3( Time-1, Period-1, Out, Rate, Libors, Mkt, Rest ).

flat3( Time, Period, In, Rate, Libors, Mkt, Value ) :-
   Time > 0, Time <= 1,
   rate( Rate, Libors, Period, LR ),
   arg( Period, Mkt, MR ),
%  approx. final market rate as full period...
   Out = In / (1 + MR/100),
%  scale final loan rate at fractional period...
   Value = Out * ( LR * Time ).

flat3( Time, Period, In, Rate, Libors, Mkt, Value ) :-
   Time > 1,
   rate( Rate, Libors, Period, LR ),
   arg( Period, Mkt, MR ),
   Out = In / (1 + MR/100),
   Value = ( Out * LR ) + Rest,
   flat3( Time-1, Period-1, Out, Rate, Libors, Mkt, Rest ).

%-----------------------------------------------------------------
% Amortized loans work only for fixed rates, and must be given
% the Start and End periods.  If these are unknown, then we cannot
% determine if the Payments are to be made in few or many periods
% with larger or smaller Payment.

amort( _, Start, End, _, _, 0 ) :- Start >= End.

amort( Principle, Start, End, fixed( Rate ), Mkt, Value ) :-
   Start < End,
   Value = Principle - Payments,
%  first compute the payment per period...
   mortgage( Start+1, End, Principle, Rate, Payment ),
%  then compute the present value of cash flows...
   amort1( Start+1, End, Payment, Mkt, Payments ).

amort1( End, End, In, Mkt, Value ) :- !,
   arg( End, Mkt, MR ),
   Value = In / ( 1 + MR/100 ).

amort1( Period, End, In, Mkt, Value ) :-
   arg( Period, Mkt, MR ),
   Out = In / ( 1 + MR/100 ),
   Value = Out + Rest,
   amort1( Period+1, End, Out, Mkt, Rest ).

% at the end, the balance of payments must be ZERO...
mortgage( End, End, In, Rate, Payment ) :- !,
   0.0 = In * ( 1 + Rate/100 ) - Payment.
```

```
mortgage( Start, End, In, Rate, Payment ) :-
   Out = In * ( 1 + Rate/100 ) - Payment,
   mortgage( Start+1, End, Out, Rate, Payment ).

%-----------------------------------------------------------------
% miscellaneous functions...

% mode(?,?,^)
append( [], L2, L2 ).
append( [ H | T ], L2, [ H | L3 ] ) :-
   append( T, L2, L3 ).

% mode(?,?,?,^)
rate( fixed( Fix ), _, _, Fix/100 ).
rate( float( Fix ), Libors, Period, Rate ) :-
   arg( Period, Libors, Float ),
   Rate = ( Float + Fix ) / 100.

% ratio of $:x where x is foreign currency...
table( yes( [
   fx(yen,0.01),
   fx(aud,0.75),
   fx(fr, 0.30),
   fx(mrk,0.50)
   ] ) ).

info( 1, info(
      market( 3.5 ),
      libor( L1, L2, L3, L4, L5, L6, L7, L8, L9, L10 ),
      const( [ Pi3 = Pi4 ] ),
      profits( [ prof( 1, Pi1, usd ),
                 prof( 2, Pi2, usd ),
                 prof( 3, Pi3, usd ),
                 prof( 4, Pi4, usd ),
                 prof( 5, Pi5, usd ),
                 prof( 6, Pi6, usd )
               ] ),
      vars(  [ Rate1, Rate2, Pi1, Pi2, Pi3, Pi4, Pi5, Pi6 ] ),
      atoms( [ rate1, rate2, pi1, pi2, pi3, pi4, pi5, pi6 ] ),
      graph([
        info( 6, 4, flat, usd,  48, 0, 10, float(0.0)   ),
        info( 3, 6, flat, usd,  48, 0, 10, float(-0.2)  ),
        info( 4, 6, flat, usd,  48, 0, 10, fixed(Rate2) ),
        info( 6, 5, flat, usd,  75, 0, 10, fixed(Rate1) ),
        info( 1, 6, one,  usd,  27, 0,  0, _ ),
        info( 6, 1, one,  aud,  38, 0,  0, _ ),
        info( 1, 6, one,  aud,  70, 0, 10, _ ),
        info( 6, 1, one,  usd,  37, 0, 10, _ ),
        info( 6, 3, one,  aud,  68, 0,  0, _ ),
        info( 3, 6, one,  aud, 130, 0, 10, _ ),
        info( 2, 5, one,  aud, 106, 0,  0, _ ),
        info( 5, 2, one,  aud, 200, 0, 10, _ ),
        info( 5, 6, one,  aud, 106, 0,  0, _ ),
        info( 6, 5, one,  aud, 200, 0, 10, _ ) ]
      ) ) ).
```